DATA AND APPLICATION MIGRATION IN CLOUD BASED DATA CENTERS: ARCHITECTURES AND TECHNIQUES

A Thesis
Presented to
The Academic Faculty

by

Gong Zhang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
August 2011

DATA AND APPLICATION MIGRATION IN CLOUD BASED DATA CENTERS: ARCHITECTURES AND TECHNIQUES

Approved by:

Professor Ling Liu, Advisor
College of Computing
Georgia Institute of Technology

Professor Edwards Omiecinski
College of Computing
Georgia Institute of Technology

Professor Shamkant B. Navathe
College of Computing
Georgia Institute of Technology

Professor Joao Eduardo Ferreira
Department of Computer Science
University of Sao Paulo

Professor Calton Pu
College of Computing
Georgia Institute of Technology

Date Approved: 10, May 2011

To my parents,

ACKNOWLEDGEMENTS

This dissertation contains substantial help and generous advise from many individuals. I would like to express my great gratitude to everyone who has contributed during the process leading to my dissertation. Here I would like to mention a few of them.

First I would like to gratefully thank the guidance and support provided by my advisor, Prof.Dr.Ling Liu during my PhD years at Georgia Institute of Technology. She has been giving me tremendous guidance and help in every aspect and every advancing footprint of my Ph.D. study, ranging from encouragement to explore new areas, to inspiration of new ideas, to honing my research capability. I am indebted for the immense help she has given me on my research work and countless hours she has spent on the technical chapters of my disseration. Overall, she set an excellent example that I can follow through my future career to become an insightful and creative researcher.

I would like to give my hearty thanks to my Ph.D. defense exam committee members: Prof.Shamkant B. Navathe, Prof.Calton Pu, Prof.Edwards Omiecinski, and Prof.Joao Eduardo Ferreira, for their continuous support and supervision starting from Ph.D. Qualification Exam, to Ph.D. Proposal Exam, to Ph.D. Oral Defense. Their constructive critiques and feedback contributed significantly to the outcome of this dissertation.

I would also like to thank my colleagues in DiSL research group for providing a dynamic study environment and for participating in valuable research discussions with me which have helped me to improve my research work. I would also like to give my sincere thanks to my collaborators at IBM T J Watson Research Center, IBM

Almaden Research Center and NEC Princeton Lab.

Most importantly, I would like to dedicate this dissertation to my father, mother and sister for their everlasting and gratuitous support for me during my Phd years. The more progress I make in my life, the more proud I feel of the values my parents have inculcated in me. It is their continuous support and warm encouragement which enabled me to enjoy every success and overcome every obstacle along the way to finish my Ph.D. degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Computing and communication have continued to impact on the way we run business, the way we learn, and the way we live. The rapid technology evolution of computing has also expedited the growth of digital data, the workload of services, and the complexity of applications. Today, the cost of managing storage hardware ranges from two to ten times the acquisition cost of the storage hardware. We see an increasing demand on technologies for transferring management burden from humans to software. Data migration and application migration are one of popular technologies that enable computing and data storage management to be autonomic and self-managing.

In this dissertation we examine important issues in designing and developing scalable architectures and techniques for efficient and effective data migration and application migration. The first contribution we have made is to investigate the opportunity of automated data migration across multi-tier storage systems. The significant IO improvement in Solid State Disks (SSD) over traditional rotational hard disks (HDD) motivates the integration of SSD into existing storage hierarchy for enhanced performance. We developed adaptive look-ahead data migration approach to effectively integrate SSD into the multi-tiered storage architecture. When using the fast and expensive SSD tier to store the high temperature data (hot data) while placing the relatively low temperature data (low data) in the HDD tier, one of the important functionality is to manage the migration of data as their access patterns are changed from hot to cold and vice versa. For example, workloads during day time in typical banking applications can be dramatically different from those during night

time. We designed and implemented an adaptive lookahead data migration model. A unique feature of our automated migration approach is its ability to dynamically adapt the data migration schedule to achieve the optimal migration effectiveness by taking into account application specific characteristics and I/O profiles as well as workload deadlines. Our experiments running over the real system trace show that the basic look-ahead data migration model is effective in improving system resource utilization and the adaptive look-ahead migration model is more efficient for continuously improving and tuning of the performance and scalability of multi-tier storage systems. The second main contribution we have made in this dissertation research is to address the challenge of ensuring reliability and balancing loads across a network of computing nodes, managed in a decentralized service computing system. Considering providing location based services for geographically distributed mobile users, the continuous and massive service request workloads pose significant technical challenges for the system to guarantee scalable and reliable service provision. We design and develop a decentralized service computing architecture, called Reliable GeoGrid, with two unique features. First, we develop a distributed workload migration scheme with controlled replication, which utilizes a shortcut-based optimization to increase the resilience of the system against various node failures and network partition failures. Second, we devise a dynamic load balancing technique to scale the system in anticipation of unexpected workload changes. Our experimental results show that the Reliable GeoGrid architecture is highly scalable under changing service workloads with moving hotspots and highly reliable in the presence of massive node failures. The third research thrust in this dissertation research is focused on studying the process of migrating applications from local physical data centers to Cloud. We design migration experiments and study the error types and further build the error model. Based on the analysis and observations in migration experiments, we propose the CloudMig system which provides both configuration validation and installation automation

which effectively reduces the configuration errors and installation complexity.

In this dissertation, I will provide an in-depth discussion of the principles of migration and its applications in improving data storage performance, balancing service workloads and adapting to the Cloud platform.

CHAPTER I

INTRODUCTION

Computing and communication have continued to impact on the way we run business, the way we learn, and the way we live. The rapid technology evolution of computing has also expedited the growth of digital data, the workload of services, and the complexity of applications. Today, the cost of managing storage hardware ranges from two to ten times the acquisition cost of the storage hardware. We see an increasing demand on technologies for transferring management burden from humans to software. Data migration and application migration are one of popular technologies that enable computing and data storage management to be autonomic and self-managing.

In this dissertation, we examine important issues in designing and developing scalable architectures and techniques for efficient and effective data migration and application migration. The first contribution we have made is to investigate the opportunity of automated data migration across multi-tier storage systems. The significant IO improvement in Solid State Disks (SSD) over traditional rotational hard disks (HDD) motivates the integration of SSD into existing storage hierarchy for enhanced performance. We developed adaptive look-ahead data migration approach to effectively integrate SSD into the multi-tiered storage architecture. When using the fast and expensive SSD tier to store the high temperature data (hot data) while placing the relatively low temperature data (low data) in the HDD tier, one of the important functionality is to manage the migration of data as their access patterns are changed from hot to cold and vice versa. For example, workloads during day time in typical banking applications can be dramatically different from those during night time. We designed and implemented an adaptive lookahead data migration model.

A unique feature of our automated migration approach is its ability to dynamically adapt the data migration schedule to achieve the optimal migration effectiveness by taking into account of application specific characteristics and I/O profiles as well as workload deadlines. Our experiments running over the real system trace show that the basic look-ahead data migration model is effective in improving system resource utilization and the adaptive look-ahead migration model is more efficient for continuously improving and tuning of the performance and scalability of multi-tier storage systems. The second main contribution we have made in this dissertation research is to address the challenge of ensuring reliability and balancing loads across a network of computing nodes, managed in a decentralized service computing system. Considering providing location based services for geographically distributed mobile users, the continuous and massive service request workloads pose significant technical challenges for the system to guarantee scalable and reliable service provision. We design and develop a decentralized service computing architecture, called Reliable GeoGrid, with two unique features. First, we develop a distributed workload migration scheme with controlled replication, which utilizes a shortcut-based optimization to increase the resilience of the system against various node failures and network partition failures. Second, we devise a dynamic load balancing technique to scale the system in anticipation of unexpected workload changes. Our experimental results show that the Reliable GeoGrid architecture is highly scalable under changing service workloads with moving hotspots and highly reliable in the presence of massive node failures. The third research thrust in this dissertation research is focused on studying the process of migrating applications from local physical data centers to Cloud. We design migration experiments and study the error types and further build the error model. Based on the analysis and observations in migration experiments, we propose the CloudMig system which provides both configuration validation and installation automation which effectively reduces the configuration errors and installation complexity.

In this dissertation, I will provide an in-depth discussion of the principles of migration and its applications in improving data storage performance, balancing service workloads and adapting to Cloud platform.

1.1 Data Migration in Multi-tiered Storage Systems

The significant IO improvements of Solid State Disks (SSD) over traditional rotational hard disks makes it an attractive approach to integrate SSDs in tiered storage systems for performance enhancement [18]. However, to integrate SSD into multi-tiered storage system effectively, automated data migration between SSD and HDD plays a critical role. In many real world application scenarios like banking and supermarket environments, workload and IO profile present interesting characteristics and also bear the constraint of workload deadline. How to fully release the power of data migration while guaranteeing the migration deadline is critical to maximizing the performance of SSD enabled multi-tiered storage system.

In order to fully capitalize on the benefits of SSDs in a multi-tiered storage system with SSDs working as the fastest tier, it is important to identify the right subset of data that needs to be placed on this tier given the limited capacity of SSD tier due to high cost per gigabyte. Specifically, we want to maximize overall system performance by placing critical, IOPS (input/output operations per second) intensive and latency-sensitive data on the fast SSD tier through two-way automated data migration between SSDs and HDDs. By working with a variety of enterprise class storage applications, we observe that many block-level IO workloads exhibit certain time-dependent regularity in terms of access patterns and temperature of extents (hot or cold). For example, in banking applications, IO workloads for account access and credit verification are typically heavier during certain hours of a day. However, such patterns may change from day-time to night-time, from day to day, from weekdays to weekends or from working days to public holidays. Thus, block-level IO profiling

is the first step for building an automated data migration system. The next big challenge is to devise strategies

In this work, we proposed an automated lookahead data migration scheme, called LAM, which aims to adaptively migrate data between different tiers to keep pace with the IO workload variations, to maximize the benefits of the fast but capacity-limited SSD tier, and to optimize the overall system performance in terms of response time and resource utilization, while limiting the impact of LAM on existing IO workloads.

More concretely, based on workload variations and temperature of block level IO access (e.g., hot or cold extents) learned through IO profiling, we predict shifts in hot-spots of block-level extents and proactively migrate those data extents whose temperature is expected to rise in the next workload into the fast SSD tier during a lookahead period. A key challenge in the LAM design is to understand and trade off multiple factors that influence the optimal lookahead migration window.

The main contributions of this work are two fold. First, we propose the need and the impact of automated deadline aware data migration through observation and analysis of IO workload scenarios from real world storage system practice. By introducing the basic data migration model in an SSD enabled multi-tiered storage system, we study the characteristics and impacts of several factors, including IO profiles, IO block level bandwidth, and the capacity of SSD tier, on improving overall performance of the tiered storage systems. Second, we propose a lookahead migration framework as an effective solution for performing deadline aware, automated data migration, by carefully managing the performance impact of data migration on existing runtime application workloads and maximizing the gains of lookahead migration. A greedy algorithm is designed to illustrate the importance of determining a near optimal lookahead window length on the overall system performance and a number of important factors, such as block level IO bandwidth, the size of SSD tier, the workload characteristics, and IO profiles. Our experiments are conducted using both the

IO trace collected from benchmarks on a commercial enterprise storage system and the simulation over the real trace. The experimental study demonstrates that the greedy algorithm based lookahead migration scheme not only enhances the overall storage system performance but also provides significantly better IO performance as compared to both basic data migration.

The efficiency of greedy algorithm based lookahead data migration is restricted by the incremental granularity and lacks flexibility. Thus an adaptive migration algorithm, which can pace with the changes of the environment of the system, is demanded. In this work, we proposed an adaptive deadline aware lookahead data migration scheme, called ADLAM, which adaptively decides the window length of lookahead based on the system parameters.

The main contributions of the data migration work are two fold. First we build a formal model to analyze the benefits of basic data migration across different phases on system response time improvements and integrate the benefits in each phase into the benefits across all the phases. Second, we present our data migration optimization process which evolves from learning phase reduction, to constant lookahead data migration and to adaptive lookahead data migration scheme. The system utility measure is proposed to compare the performance gains in each data migration model. We propose an adaptive lookahead migration approach, which works as an effective solution for performing deadline aware data migration by carefully trading off the performance gains achieved by lookahead migration on the next workload and the potential impacts on existing workloads. This approach centers around a formal model which computes the optimal lookahead length by considering a number of important factors, such as block level IO bandwidth, the size of SSD tier, the workload characteristics, and IO profiles. Our experiments confirm the effectiveness of the proposed adaptive data migration scheme by testing the IO traces collected from benchmark and commercial applications running on an enterprise multi-tiered storage server. The experiments

show that ADLAM not only improves the overall storage performance, but also outperforms the basic data migration model and constant lookahead migration strategies significantly in terms of system response time improvements.

1.2 Service Workload Migration for Reliability and Load Balance

Although distributed system increases the scalability though distributing the workload among different participating nodes, however the heterogeneity among nodes presents a significant challenge towards system load balance and reliability. Workload migration permitting replicas, or replication, is an effective approach to achieve load balance and reliability. We show the power of replication in an overlay network based distributed location service network, GeoGrid Network. In contrast to centralized client-server architecture, decentralized management and provision of distributed location based services have gained lot of attention due to its low cost in ownership management and its inherent scalability and self configurability.

Measurements [110, 68] performed on deployed overlay networks show that node characteristics such as availability, capacity and connectivity, present highly skewed distribution and such inherent dynamics creates significant variations, even failures, on the services provided by the overlay systems. For example, a sudden node failure that causes the service interruption may lead the system to exhibit dramatic changes in service latency or return inconsistent results. Furthermore, increasing population size of mobile users and diversity of location-based services available to mobile users have displayed rapidly changing user interests and behavior patterns as they move on the road, which creates moving hot spots of service requests and dynamically changing workloads. Thus an important technical challenge for scaling location service network is to develop a middleware architecture that is both scalable and reliable, on top of a regulated overlay network with node dynamics and node heterogeneity, for large scale location based information delivery and dissemination. By scalable, we mean that

the location service network should provide effective load balancing scheme to handle the growing number of mobile users and the unexpected growth and movements of hot spots in service demand. By reliable, we mean that the location service network should be resilient in the presence of sudden node failures and network partition failures.

In this work we proposed Reliable GeoGrid, a decentralized and geographical location aware overlay network service architecture for scalable and reliable delivery of location based services. The main contributions of this work are two fold. First, we describe a distributed replication scheme which enables the reliable location service request processing in an environment of heterogeneous nodes with continuously changing workloads. Our replication framework provides failure resilience to both individual node failures and massive node failures, aiming at keeping the service consistently accessible to users and eliminating the sudden interruption of the ongoing tasks. Second, we present a dynamic replica-based load balancing technique, which utilizes a parameterized utility function to control and scale the system in the presence of varying workload changes by taking into account of several workload relevant factors. Our experimental evaluation demonstrates that Reliable GeoGrid architecture is highly scalable under changing workloads and moving hotspots, and highly reliable in the presence of both individual node failures and massive node failures.

1.3 Application Migration

The last piece of my disseration work is dedicated to application migration from local data center to the Cloud. Cloud infrastructures such as EC2, App Engine provide a flexible, economic and convenient solution for enterprise applications through the pay-as-you-go business model. More and more enterprises are planning to migrate their services to Cloud platform. However, migrating service to Cloud turns out to be a complicated, expensive, error prone process because of many reasons such

as hardware errors, software errors, configuration errors, access permission errors, performance anomalies and etc. Moreover,enterprise applications consist of massively distributed networked components, and sophisticated dependency and logics exist among system components, which further aggravates the complexity and hardness of the migration problem. However, there exists neither a framework nor a tool to simplify the migration process and validate if the system is working correctly after the migration. Most of the literature is focusing on improving the efficiency of migrating single virtual machine images. I will dedicate the 4th Chapter of this dissertation to the migration configuration validation and system installation automation. In this work, I design an effective policy based migration configration validation framework which simplifies the application migration process and reduces the probability of errors. We argue that such framework development can significantly facilitate the migration process and validate the effectiveness of application migration.

1.4 Contribution of the Dissertation

The contribution of this dissertation is that we study important research problems in data and application migration and further propose the development of the technical solutions for the challenges. The first contribution we have made is to investigate the opportunity of automated data migration across multi-tier storage systems. The significant IO improvement in Solid State Disks (SSD) over traditional rotational hard disks (HDD) motivates the integration of SSD into existing storage hierarchy for enhanced performance. We developed adaptive look-ahead data migration approach to effectively integrate SSD into the multi-tiered storage architecture. When using the fast and expensive SSD tier to store the high temperature data (hot data) while placing the relatively low temperature data (low data) in the HDD tier, one of the important functionality is to manage the migration of data as their access patterns are changed from hot to cold and vice versa. For example, workloads during day time

in typical banking applications can be dramatically different from those during night time. We designed and implemented an adaptive lookahead data migration model. A unique feature of our automated migration approach is its ability to dynamically adapt the data migration schedule to achieve the optimal migration effectiveness by taking into account of application specific characteristics and I/O profiles as well as workload deadlines. Our experiments running over the real system trace show that the basic look-ahead data migration model is effective in improving system resource utilization and the adaptive look-ahead migration model is more efficient for continuously improving and tuning of the performance and scalability of multi-tier storage systems. The second main contribution we have made in this dissertation research is to address the challenge of ensuring reliability and balancing loads across a network of computing nodes, managed in a decentralized service computing system. Considering providing location based services for geographically distributed mobile users, the continuous and massive service request workloads pose significant technical challenges for the system to guarantee scalable and reliable service provision. We design and develop a decentralized service computing architecture, called Reliable GeoGrid, with two unique features. First, we develop a distributed workload migration scheme with controlled replication, which utilizes a shortcut-based optimization to increase the resilience of the system against various node failures and network partition failures. Second, we devise a dynamic load balancing technique to scale the system in anticipation of unexpected workload changes. Our experimental results show that the Reliable GeoGrid architecture is highly scalable under changing service workloads with moving hotspots and highly reliable in the presence of massive node failures. The third research thrust in this dissertation research is focused on studying the process of migrating applications from local physical data centers to Cloud. We design migration experiments and study the error types and further build the error model. Based on the analysis and observations in migration experiments, we propose the CloudMig

system which effectively reduces the configuration errors and installation complexity through providing both configuration validation and installation automation.

1.5 Organization of the Dissertation

The major part of the dissertation is organized into 3 parts. First, we will inroduce data migration in multi-tiered storage system. We introduce the motivation behind the research problem and propose our lookahead data migration. Also we build the theorectical model which can compute the lookahead window length optimally. Next we introduce the service migration in decentralized systems, that is, we discuss the reliability and load balance prolems in large scale distributed systems and then discuss our proposed hybrid replication scheme which can handle both the reliability and load balance problem. Finally, we introduce our latest work on application migration. We discuss the motivation experiments we have done and the keys to solve this prolem and our policy based solution in solving this problem. Finally, we conclude this dissertation.

CHAPTER II

DATA MIGRATION IN MULTI-TIERED STORAGE BASED CLOUD ENVIRONMENT

2.1 Introduction

The rise of solid-state drives (SSDs) in enterprise data storage arrays in recent years has put higher demand on automated management of multi-tiered data storage software to better take advantage of expensive SSD capacity. It is widely recognized that SSDs are a natural fit for enterprise storage environments, as their performance benefits are best leveraged for enterprise applications that require high inputs/outputs per second (IOPS), such as transaction processing, batch processing, query or decision support analysis [93, 98].

Recently, a number of storage systems supporting Flash devices (SSDs and memory) have appeared in the marketplace such as NetApp FAS3100 system [7], IBM DS8000[5] and EMC Symmetrix [3]. Most of these products fall into two categories in terms of the ways of integrating Flash devices into the storage system. The first category involves the products that have taken the approach of utilizing Flash-memory based caches to accelerate storage system performance [7]. The main reason that is in favor of using Flash devices as cache includes the simplicity of integration into existing systems without having to explicitly consider data placement and the performance improvement by increasing cache size at lower cost (compared to DRAM) [10]. The second category includes those vendors that have chosen to integrate SSDs into tiered storage architectures as fast persistent storage [5]. The arguments for using and integrating flash devices in an SSD form factor as persistent storage in a tiered system include issues such as data integrity, accelerated wear-out and asymmetric

write performance.

Given the limited capacity of SSD tier in the multi-tiered storage systems, it is critical to place the most IOPS-intensive and latency sensitive data on the fastest SSD tier, in order to maximize the benefits achieved by utilizing SSD in the architecture. However, a key challenge in addressing two-way data migration between SSDs and HDDs arises from the observation that hot-spots in the stored data continue to move over time. In particular, previously cold (i.e., infrequently accessed) data may suddenly or periodically become hot (i.e., frequently accessed, performance critical) and vice versa. Another important challenge for automated data migration is the capability to control and confine migration overheads to be within the acceptable performance tolerance of high priority routine transactions and data operations.

By analyzing the real applications in environments such as banking settings and retail stores with a comercial enterprise multi-tiered storage server, one can discover certain temporal and spatial regularity in terms of access patterns and temperature of extents. For example, in a routine banking environment, the major workload during the daytime centers around transaction processing and thus certain indices become hot in the daytime period, while at the night time, the workload switches into report generation type and correspondingly different indices become hot. Generally speaking, such patterns may not only change from day-time to night-time and it is also likely that the pattern changes from day to day, from weekdays to weekends or from working period to vacation period. This motivates us to utilize IO profile as a powerful tool to guide the data migration. Further, in order to improve system resource utilization, the data migration techniques devised for multi-tiered storage architecture need to be both effective and non-intrusive. By effective, we mean that the migration activity must select an appropriate subset of data for migration, which can maximize overall system performance while guaranteeing completion of the migration before the onset of the new workload (i.e., within a migration deadline). By non-intrusive, we

mean that the migration activity must minimize its impact on concurrently executing storage workloads running over the multi-tier storage system.

In this chapter, we present an adaptive deadline aware lookahead data migration scheme, called ADLAM, which aims to adaptively migrate data between different storage tiers in order to fully release the power of the fast and capacity-limited SSD tier to serve hot data and thus improves system performance and resource utilization and meanwhile limiting the impact of ADLAM on existing active workload. Concretely, we want to capitalize on the expensive SSD tier primarily for hot data extents based on IO profiling and lookahead migration. IO profiling enables the shifts in hot spots at the extent level to be predictable and exploitable, and based on this lookahead migration practively migrates those extents, whose heat is expected to rise in the next workload, into SSD tier during a lookahead period. To optimally set the lookahead migration window, multiple dimensions that impact the optimal lookahead length needs to be exploited. By optimal we mean that this lookahead migration should effectively prepare the storage system for the next workload by reducing the time taken to achieve the peak performance of the next workload and maximizing the utilization of SSDs. We deployed a prototype ADLAM in an operational enterprise storage system and the results show that lookahead data migration effectively improves system response with reduced average cost by fully utilizing the scarce resource such as SSDs.

The main contributions of this chapter are two fold. First, we describe the needs and impacts of the basic deadline aware data migration on improving system performance from real storage practice. We build a formal model to analzye the benefits of basic data migration across different phase on system response time improvements and integrates the benefits in each phases into the benefits across all the phases.

Second, we present our data migration optimization process which evolves from

learning phase reduction, to constant lookahead data migration and to adaptive lookahead data migraiton scheme. The system utility meansure is built to compare the performance gains in each data migration model. Adaptive lookahead migration apparch, which works as an effective solution for performing deadline aware data migration by carefully trading off the peformance gains achieved by lookahead migration on the next workload and the potential impacts on existing workloads. This approach centers around a formal model which computes the optimal lookahead length by considering a number of important factors, such as block level IO bandwidth, the size of SSD tier, the workload characteristics, and IO profiles. Our experiments confirms the effectiveness of the proposed adaptive data migration scheme by testing the IO traces collected from benchmark and commercial applications running on an enterprise multi-tiered storage server. The experiments shows that ADLAM not only improves the overall storage performance, but also outperforms the basic data migration model and constant lookahead migration strategies significantly in terms of system response time improvements.

2.2 ADLAM *Overview*

A typical SSD-based multi-tier storage system architecture consists of three layers. The bottom layer contains many physical disks classified into different tiers according to the IO access performance characteristics, (for example as the figure shows SSD provides the high speed IO access as tier 0, and SATA disks and SCSI disks work as tier 1 to provide mass storage), and the storage controller in the middle layer is response for the functionalities such as generating data placement advise, planning data migration and providing device virtualization. As the top layer, virtual LUN presents hosts a logical view of the storage system virtualy comprised of many LUNs and each LUN is further divided into many logical extents. The storage controller manages the mapping between logical extents and the physical extents in the physical

layer. The two way data migration between tiers presents different IO features. Data migration from tier 1 to tier 0 achieves higher access speed in the price of higher per GB access cost, while the reverse data migration decreases IO speed and IOPS correspondingly. Two way data migration among different tiers provides us the flexibility to schedule the resource to meet different access speed demand and cost on the data extent level.

2.2.1 Motivation Scenario and Assumptions

In this chapter, we focus on the working environments like banking scenarios or retail stores where IO workloads typically alternate between periods of high activities and low activities and also the workload characteristics may change from daytime to night time activities. For example, in a typical banking environment, the workload during the daytime (business hours) primarily focuses on the transaction processing tasks which create intense accesses to indices and some small set of data out of the large data collection, such as access permission logs. However, during the night time, the workload type is often switched into report generating style and the high access frequency data is also changed into different indices or different logs. Every 12 hours, one workload finishes its cycle and the next workload starts its cycle. Every 24 hours, a complete cyclic period of workload pattern (with respect to two workloads in this example) finishes and the next period starts. Apparently, data segments like indices and access logs attract significantly dense IO accesses and are hot. Improving the access speed for these "hot data" is highly critical to the performance enhancement of the entire storage system. Through our experiences working with a number of enterprise class storage system applications in banking and retail store industries, we observe that many applications have similar IO workloads as those in banking or retail stores and exhibit similar time-dependent patterns in terms of random IO accesses and batch IO reads.

We run the Industry standard IO benchmark SPC1 and TPCE for a duration of time on a proprietary enterprise storage system respectively. Certain stability for a given workload presented by hot data during its running cycle is confirmed. The experimental results reveals two interesting observations. First the hot extents bands for the two different workloads are totally different. Second, stability holds on both hot data and cold data with given workload active cycle for both SPC1 and TPCE workloads. Readers may refer to our technical report for further details on this analysis.

The high disparity of access speed between traditional HDDs and SSDs and the highly cyclic behaviors of different IO workloads motivate us to migrate the hot data like indices into SSD for each workload type and thus significantly improve the access speed for hot data, which is a known bottleneck that dominates the storage system performance. Another important requirement for devising data migration schedule is the fact that the workload pattern tends to alternate, for example between daytime and nighttime, and thus different workload cycles generate different hot data and hence it is compulsory to finish the desired data migration before the deadline of the corresponding workload to which the hot data is associated. To address the data migration problem that is constrained by the workload deadline, we identify the following assumptions for the adaptive deadline aware data migration problem:

1. The IO workloads exhibit cyclic behaviors and the set of workload types alternate between one another within a given period, such as every 24 hours.

2. The data access patterns, especially the hot data extents, can be learned by continuously monitoring the data heat.

3. Fast storage resource like SSD exhibits scarcity in terms of per unit cost.

4. Data migration for each aworkload is bounded by certain deadline.

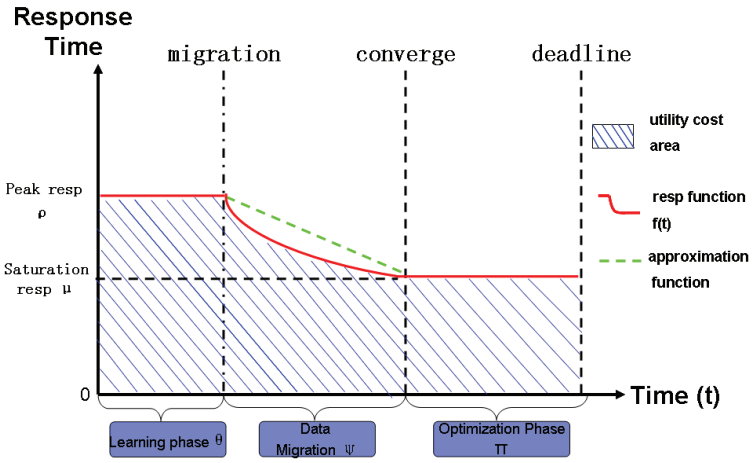

Figure 1: Migration Computation Model

Bearing with these constraints, we develop an adaptive deadline-aware lookahead data migration framework, – ADLAM, aiming at optimizing system resource utilization by adaptively migrating data between faster storage tier and slow storage tier triggered by workload alternation while guaranteeing the deadline requirements.

2.2.2 Storage Utility Cost

In multi-tier storage system, response time (for abbreviation,we refer response time as "resp") is the major performance goal whose optimization is concerned the most by the users. Thus, the principal performance optimization objective using data migration is to reduce the response time as the storage system runs. We first define system response time function as follows:

Definition 1. A function recording the response time during a time period from $t1$ to $t2$ for the multi-tier storage system is called *"response time function"* and is denoted as $f(t), t \in [t1, t2]$.

Definition 2. The system utility cost, denoted by U, is the summation of the response time of the multi-tier storage system at each time t over the entire time period of $[t1, t2]$ and $t \in [t1, t2]$. That is, $U_{t_1,t_2}(t) = \int_{t1}^{t2} f(t)dt$.

Our goal is to minimize the system utility cost while guaranteeing the migration deadline. If migration is not employed, then the system response time is maintained at its average peak response time, denoted by ρ.

Definition 3. Let ϕ denote a workload duration (cycle) and U_0 denote the utility cost of the system when data migration is not equipped and $U_0 = \rho \times \phi$. We call the utility improvement from utility cost under a system without data migration to improved utility cost with a system powered by data migration the *utility rise* denoted by ΔU_0, and we have $\Delta U_0 = U_0 - U$.

2.2.3 Basic Migration Model

Basically, the migration process can be divided into three phases based on its impact on the overall response time of the system.

(1) Workload learning phase: The primary task in this phase is continuously monitoring the IO profile and sorting the extent list on the order of heat. To ensure data collection process shed ignorible impacts on the ongoing workload, very limited resource is allocated by this process. As shown in Figure 1, if the learning phase takes θ time, then the utility cost of this phase is $U = \int_0^{\theta} f(t)dt$, which can be approximated by $\rho \times \theta$.

(2) Data migration: In this phase, hot data extents are migrated from slow HDD tier to fast SSD tier.

In this phase, the extents are migrated on the heat order of data extents. Because there are limited number of hot expents which plays significant impacts on the response time, thus, the migration impacts on the response time observe the law of "diminishing marginal returns". That is, as the heat of the migrated extent decreases, each additional data extent migrated yields smaller and smaller reduction in response time and finally at some time point, the migration impact is saturated and no reduction in response time can be achieved by further migration. We call this time point

the "convergence point". As shown in Figure 1, if the data migration phase takes ψ time, then the utility cost of this phase is $\int_{\theta}^{\theta+\psi} f(t)dt$.

(3) Optimization Phase:

After "convergence point" arrives, system enters into the "optimization phase", and the response time of the ongoing workload maintains at the stable minimum level until the active workload reaches its deadline and finishes its running cyle. As shown in Figure 1, if the optimization phase takes π time, then the utility cost of optimization phase is $\int_{\theta+\psi}^{\theta+\psi+\pi} f(t)dt$. Figure 1 illustrates the three phases that each workload experiences.

2.2.4 Utility Cost of Single Data Migration

As illustrated in the basic data migration model, individual workload experiences three phases in one cycle and for simplicity, we call such basic data migration process for an individual workload as *single migration*. The utility cost incurred in this process forms a basic unit in computing the overall system utility cost. In this section, we analyze the utility cost for single migration and further build the model to computing the utility cost.

For an individual workload, the workload cycle time ϕ equals to the sum of the time length of three phases, i.e., $\phi = \theta + \psi + \pi$. Thus, the aggregation of the utility cost in each phase, corresponding to the solid line shaded area in Figure 1, forms the utility cost of the workload over one cycle, as shown in the below lemma.

Lemma 1. The utility cost of an individual workload over one cycle ϕ is:

$$\begin{aligned} U &= \int_{0}^{\phi} f(t)dt \\ &= \int_{0}^{\theta} f(t)dt + \int_{\theta}^{\theta+\psi} f(t)dt + \int_{\theta+\psi}^{\theta+\psi+\pi} f(t)dt. \end{aligned} \tag{1}$$

Data migration time refers to the time duration length, χ, taken to migrate the data to SSD until it is full. For SSD tier with capacity C and bandwidth B allocated

for data migration, then the total time duration length that is needed to fill in the SSD tier is $\chi = \frac{C}{B}$. We define "saturation time" λ as the time duration from the beginning of data migration to the time point when the impacts of migration on response time reduction is saturated or convergence point (see section 2.2.3) is reached. Depending on inherent SSD properties and the allocated migration bandwidth, saturation time may be longer or shorter than data migration time and it depends on specific applications and SSDs. For simplicity, data migration time is used as an reasonable approximation for saturation time, i.e., $\lambda = \chi$.

With the peak response time ρ which is achived when data migration is not employed and the saturation response time μ, the response time function $f(t)$ in data migration phase is approximated with a linear function $f(t) = \frac{\mu-\rho}{\theta} \times t + \rho$, $t \in [\theta, \theta + \psi]$,which is shown as green dotted line above response time funtion curve in Figure 1. Thus, the response time function for single migration is:

$$f(t) = \begin{cases} \rho & t \in [0, \theta] \\ \frac{\mu-\rho}{\psi} \times t + \rho & t \in [\theta, \theta + \psi] \\ \mu & t \in [\theta + \psi, \phi] \end{cases} \tag{2}$$

From equation 1 and equation 2, the utility cost for a single migration process is derived as follows:

Theorem 1. The utility cost of a single migration is approximated as:

$$U = \int_0^{\phi} f(t)dt = (\rho - \mu) \times (\frac{1}{2} \times \psi + \theta) + \mu \times \phi \tag{3}$$

And the utility rise over the baseline case in which no migration is employed is:

$$\begin{aligned} \Delta U_0 &= \rho \times \phi - ((\rho - \mu) \times (\frac{1}{2} \times \psi + \theta) + \mu \times \phi) \\ &= (\rho - \mu) \times (\phi - \theta - \frac{1}{2} \times \psi) \end{aligned} \tag{4}$$

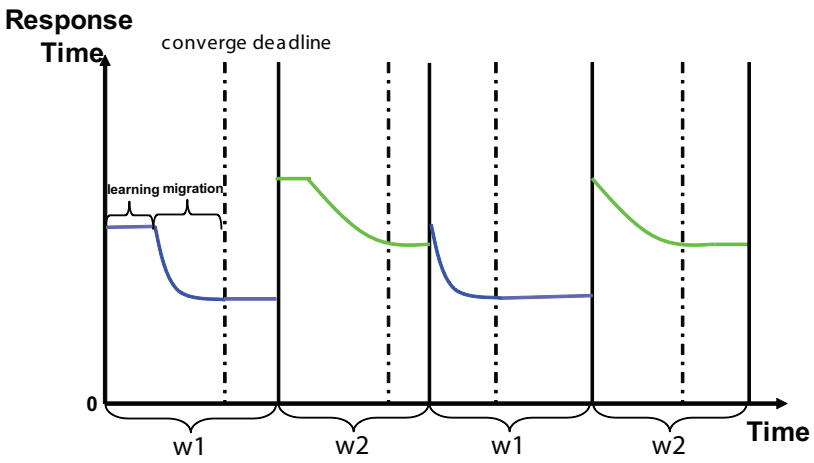

Figure 2: Workload cycles with learning phase reduced

2.2.5 Migration Model with Reduced Learning Phase

By utilizing the data heat stability discussed earlier, one can reuse the stable IO profile in subsequent workflow cycles and thus reduce the learning phase significantly. Concretely, one can turn on the learning phase in the beginning workload cycles. Once the data heat is stabilized, the learning phase can be reduced for the subsequent workload cycles by reusing the data heat information learned at the beginning rounds of workload cycles. The reduction of learning cycles cuts off the redundant learning cost and enables the system to enter the optimization phase earlier than the basic migration model. In Figure 2, the first run of w1 and w2 does not eliminate the learning phase and in the second run the learning phase is eliminated.

The corresponding response time function for learning phase reduced data migration process is:

$$f(t) = \begin{cases} \frac{\mu-\rho}{\psi} \times t + \rho & t \in [0, \psi] \\ \mu & t \in [\psi, \phi] \end{cases} \tag{5}$$

From equation 1 and equation 2, the utility cost for a learning phased reduced migration process is derived as follows:

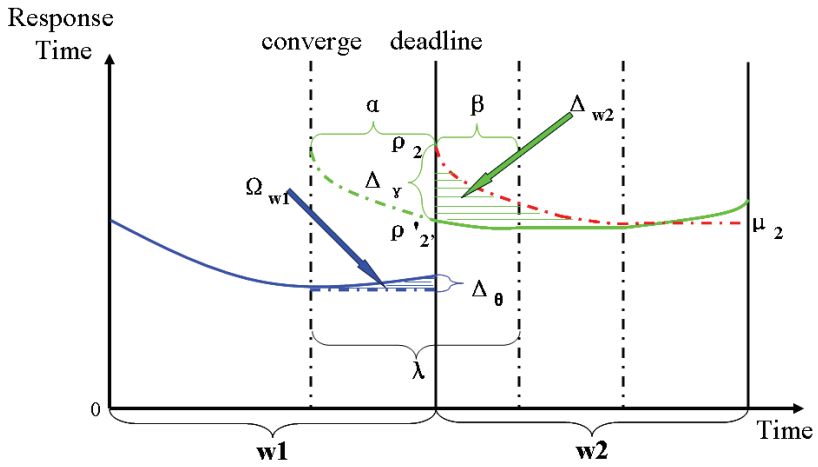

Figure 3: Lookahead migration

Theorem 2. The utility cost of a reduced learning data migration is as:

$$U = \int_0^{\phi} f(t)dt = (\rho - \mu) \times \frac{1}{2} \times \psi + \mu \times \phi \tag{6}$$

And the utility rise over the baseline case in which no migration is employed is:

$$\begin{aligned} \Delta U_0 &= (\rho - \mu) \times \frac{1}{2} \times \psi + \mu \times \phi - \rho \times \phi \\ &= (\rho - \mu) \times (\phi - \frac{1}{2} \times \psi) \end{aligned} \tag{7}$$

2.3 Adaptive Lookahead Migration

2.3.1 Lookahead Migration

Adaptive lookahead migration is a further improvement over learning phase reduced data migration model. As shown in Figure 3. For a configurable lookahead window length α, the solid green line and the dashed green line in $w2$ workload time together show the effects of migration on response time of workload $w2$. The dashed red curve shows the "virtual" situation on how $w2$ response time evolves if no lookahead data migration is deployed. Similary, the migration effects under α lookahead on $w1$ is shown with the blue solid curve and blue dashed curve in $w1$ workload time. α indicates that data migration for $w2$ starts α units of time ahead of the time

point when the $w1$ workload finishes its running cycle. The power force of lookahead migration is rooted from the the fact that after $w1$ enters into its convergence point, further migration of $w1$ extents creates ignorible benefits on system response time. Thus it is more cost-effective to start migrating the hottest extents of $w2$ into SSD tier ahead of activating time of $w2$.

In practice, because the capacity of SSD is limited, it is very likely that the SSD has already been filled with $w1$ extents when lookahead migration starts. Thus typically, lookahead migration incurs the swapping process in which relatively cold $w1$ extents are replaced by the hottest $w2$ extents. Such swapping increases the response time of ongoing workload $w1$ to some level, which is called "bending effect", as shown in Fig. 3, the solid blue response time curve of $w1$ is slightly bending upward when lookahead migration of $w2$ starts. The dashed green line in $w2$ cycle essentially depicts the implicit data migration employed for $w2$ and we can see that because of such implicit data migration is executed in advance, the peak response time of $w2$ workload experienced by the client applications by the amount of Δ_γ. The implicit undergoing lookahead migration also reduces the time length taken to reach the convergence point perceived by the clients from $(\alpha + \beta)$ in the case of no lookahead migration to β under the lookahead migration for workload $w2$.

The shaded green area, denoted by Δ_{w2} represents the resource utilization gain achieved by lookahead migration in $w2$, and the shaded blue area, denoted by Ω_{w1}, indicates the resource utilization loss experienced by $w1$. As long as the gain by Δ_{w2} is greater than the loss of Ω_{w1}, i.e., $\Delta_{w2} > \Omega_{w1}$ holds, lookahead migration with window size α can benefit the system utilizations in a cost-effective manner. Obviously, when difference of Δ_{w2} and Ω_{w1} is maximized, the corresponding looahead length releases its full power and lookahead data migration achieves its maximum effects in term of system utility optimization. The quesiton becomes how to decide the optimal lookahead length. In [135], we proposed a greedy algorithm, which can

compute a near optimal lookahead length depending on the granularity used. Next, we build a formal model on the system utility optimization using lookahead migration and derive the approach to compute the optimal lookahead length precisely.

2.3.2 Computing Optimal Lookahead Migration Window Size

In this section, we design an algorithm which computes the near optimal lookahead window size by taking into account several important factors, such as the IO bandwidth, the size of SSD tier (number of SSDs), and IO workload characteristics. Let $w1$ denote the current dominating workload running in the system and $w2$ denote the next workload that will dominate the IO access after $w1$ approaches the end. We determine the optimal lookahead migration window size in the following three steps.

2.3.2.1 Finding the utility improvement of workload $w2$

As shown in Figure 3, similarly we can apply a linear function to approximate the response time function for workload $w2$ in the migration function and thus derive the following function to depict the response time curve.

$$f(t) = \begin{cases} \frac{\mu_2 - \rho_2}{\psi_2} \times (t + \alpha) + \rho_2 & t \in [0, \beta] \\ \mu_2 & t \in [\beta, \phi_2] \end{cases} \tag{8}$$

In Figure 3, ρ_2 is the peak response time for original response time curve without employing lookahead migration for $w2$.

Theorem 3. The utility gain of $w2$ through using α lookahead migration compared with purely learning phase reduced migration is

$$\begin{aligned} \Delta_{w2} &= U_2(0) - U_2(\alpha) \\ &= (\rho_2 - \mu_2) \times \alpha - \frac{\rho_2 - \mu_2}{2 \times \psi_2} \times \alpha^2 \end{aligned} \tag{9}$$

2.3.2.2 Finding the utility loss of workload $w1$

As shown in Figure 3, starting migration of $w2$ workload causes the increment of response time of workload $w1$ and the peak response time in this increment process happens at the deadline of workload $w1$, which is called *reverse peak response time* η. The reverse response time increment process can be approximated by function $f\prime(t) = \frac{\rho_1-\mu_1}{\psi_1} \times t + \mu_1$. Let $t = \alpha$, we derive $\eta = f\prime(\alpha) = \frac{\rho_1-\mu_1}{\psi_1} + \mu_1$. Thus, the performance loss of workload $w1$ is as follows:

Theorem 4.

$$\Omega_{w1} = \frac{\rho_1 - \mu_1}{2 \times \psi_1} \times \alpha^2 \tag{10}$$

2.3.2.3 Compute the near optimal lookahead window size

The lookahead migration utility can be derived from the performance gain of $w2$ and performance loss of $w1$ using the following theorem.

Theorem 5. The utility of α lookahead migration,

$$\begin{aligned} \Gamma(\alpha) &= \Delta_{w2} - \Omega_{w1} \\ &= (\rho_2 - \mu_2)\alpha - \frac{\rho_2 - \mu_2}{2 \times \psi_2}\alpha^2 - \frac{\rho_1 - \mu_1}{2 \times \psi_1}\alpha^2 \\ &= (\rho_2 - \mu_2)\alpha - (\frac{\rho_2 - \mu_2}{2 \times \psi_2} + \frac{\rho_1 - \mu_1}{2 \times \psi_1})\alpha^2 \end{aligned} \tag{11}$$

As our goal is to maximize $\Gamma\alpha$ and its maximum value is achieved at $\alpha = \frac{\psi_1\psi_2(\rho_2-\mu_2)}{\phi_1(\rho_2-\mu_2)+\phi_2(\rho_1-\mu_1)}$ and the maximum lookahead utility is $\Gamma_{max} = \frac{3\phi_1\phi_2(\rho_2-\mu_2)^2}{2\psi_1(\rho_2-\mu_2)+2\psi_2(\rho_1-\mu_1)}$. With this approach to compute the optimal lookahead length, even the system parameter changes, such as migration bandwidth or SSD sizes changes, a new lookahead length can be adaptively recomputed and thus guarantee the adaptivity of the proposed approach.

2.4 Experimental Evaluation

This section evaluates the performance of our adaptive, deadline-aware lookahead migration scheme through four sets of experiments.

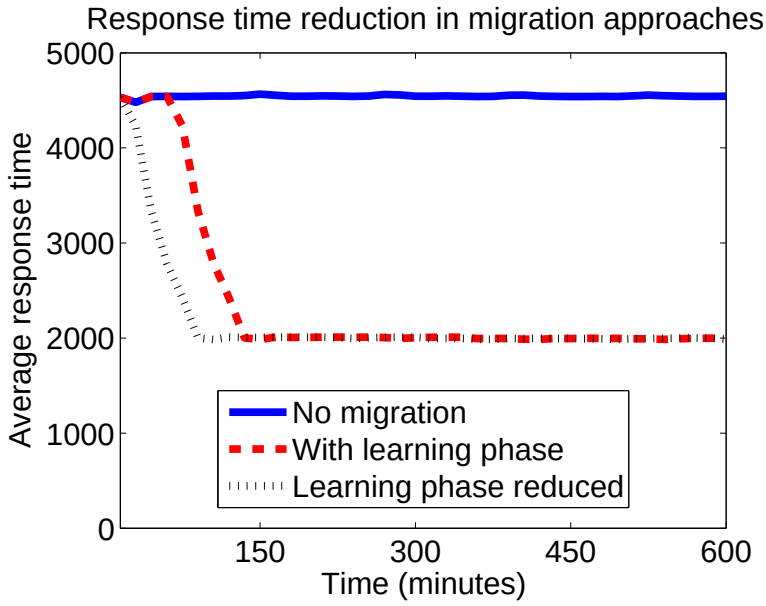

Figure 4: Response time evolvement in scenarios without migration, with normal migration and with learning phase reduced migration approach in TPCE trace with 2 SSDs and 32 HDDs and with 4GB/5mins migration bandwidth allocated

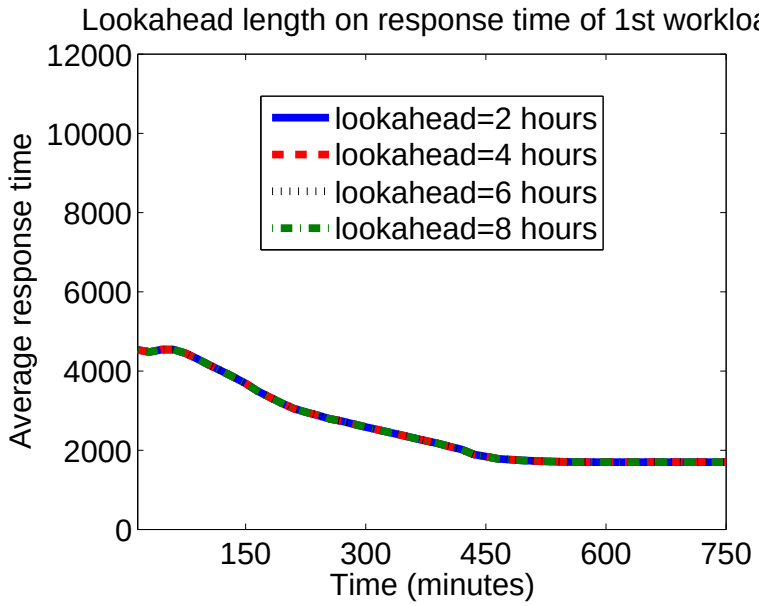

Figure 5: The impacts of lookahead length on the first workload (TPCE trace on cluster 2) when 2 $\leq$lookahead length$\leq$ 8 and migration bandwidth=2GB/5mins in the TPCE-SPC1 sequential workload with 3 SSDs and 32 HDDs

To examine the benefits of lookahead migration, we first collected IO trace from real enterprise storage controller and implemented the lookahead data migration

scheme in a simulator that simulates the real enterprise storage system. We developed a trace driven simulator, which consists of a storage unit with dual SMP server processor complexes as controllers, sufficiently large memory with large size cache, multiple types of disk drives (including SATA, SCSI and SSD), multiple FICON or ESCON adapters connected by a high bandwidth interconnect, and management consoles. Its design is to optimize both exceptional performance and throughput in supporting critical workloads and around the clock service.

Our simulator simulates the IO processing functionalities of the enterprise storage systems driven by the IO pressure trace collected from running benchmark workloads on the real enterprise storage system. For the experiments in this chapter, the simulator simulates a multi-tiered storage hierarchy consisting of SSD drives as tier 0 and SATA drives as tier 1 and the total physical capacity is up to hundreds of terabytes in terms of hundreds of disk drives. For the experiments in this chapter, we run the simulations on a server machine with 4 processors and 12 GB memory running Linux Fedora 10.

IO traces for the simulator were generated by running the Industry standard IO benchmark SPC1 and TPCE on the referred proprietary enterprise storage system for a duration about 24 hours. In all the following experiment, "Average response time" refers to the average of the accumulative response time of all the requests for a extent in the extents pool at a particular time point and it is in the unit of miliseconds.

2.4.1 Effect of Basic Data Migration

Using the IO trace collected from running the benchmarks on real enterprise storage system to the simulator, we measured the average response time across all data extents every 15 minutes (1 cycle). We compare three different approaches: no data migration functionality, the basic data migration scheme described in Section 2.2.3 and reducing the learning phase by reusing the IO profile. Figure 4 shows the comparison of

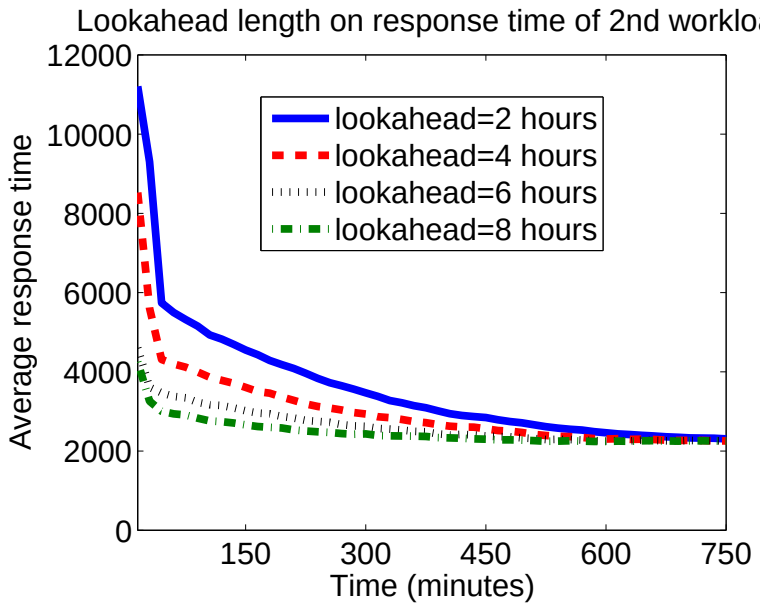

Figure 6: The impacts of lookahead length on the second workload (SPC1 trace on cluster 1) when 2 $\leq$lookahead length$\leq$ 8 and migration bandwidth=2GB/5mins in the TPCE-SPC1 sequential workload with 3 SSDs and 32 HDDs

system response time with the three basic migration approaches under the IO trace collected from the TPCE benchmark. The results show that the average response time without data migration is more than twice the response time achieved by the other two approaches which perform basic data migration. The response time remains flat because TPCE workload presents even distribution over time interval. When we turn on the basic data migration scheme, in the first 1 hour, the IO profiles of extents are collected and the extents are sorted based on their heat levels. Then the basic data migration scheme is activated and the extents are migrated in the descending heat order. During the first hour learning phase, the system response time maintains at the same level as the case without data migration. After the first hour learning phase, as the data migration process begins, the system response time starts to drop. Within a few cycles, the data migration process reaches the convergence point and enters into the optimization phase as further data migration no longer creates notable improvement on response time. The graph also shows that reusing IO profiles for a

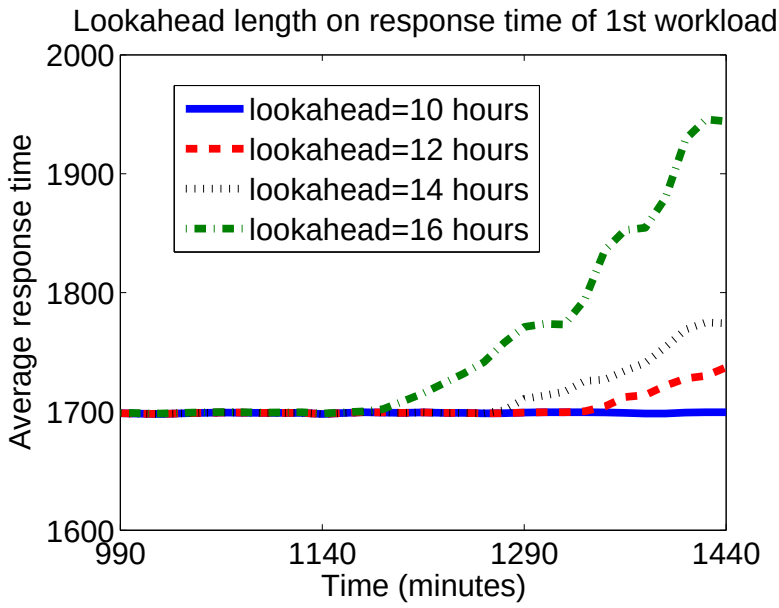

Figure 7: The impacts of lookahead length on the first workload (TPCE trace on cluster 2) when 10 $\leq$lookahead length$\leq$ 16 and migration bandwidth=2GB/5mins in the TPCE-SPC1 sequential workload with 3 SSDs and 32 HDDs

workload with stable characteristics can improve performance at a faster rate than an approach which independently learns the IO profiles each time.

2.4.2 Effect of Look-ahead Window Length

In this set of experiments, we study the effects of lookahead window length on data migration. The storage layer is configured with 3 SSDs and 33 HDDs. The total capacity of SSD is set to be much less than the capacity of HDD. The input trace is a hybrid trace with TPCE trace as the first half and SPC1 trace as the second half and works as workload $w1$ and workload $w2$ respectively. In order to understand the impacts of lookahead window on system performance, we vary lookahead window length from 1 hour to 16 hours. For each window size increment, we output the system wide average response time every 15 minutes to observe the impacts of lookahead migration on the performance of workload $w1$ and $w2$. To simplify the problem, we isolate the lookahead migration only to workload $w2$ which help to extract segment that is repeated among the continuous workloads and center on the impacts of lookahead

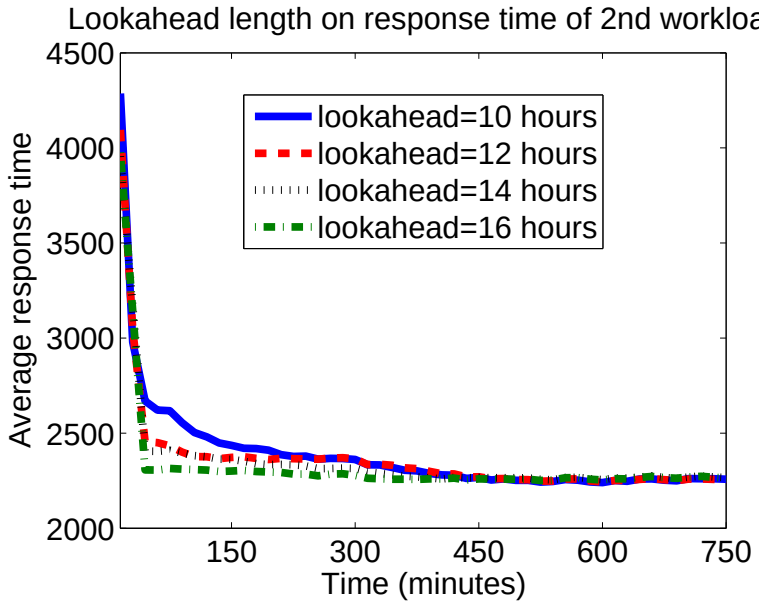

Figure 8: The impacts of lookahead length on the second workload (SPC1 trace on cluster 1) when 10 $\leq$lookahead length$\leq$ 12 and migration bandwidth=2GB/5mins in the TPCE-SPC1 sequential workload with 3 SSDs and 32 HDDs

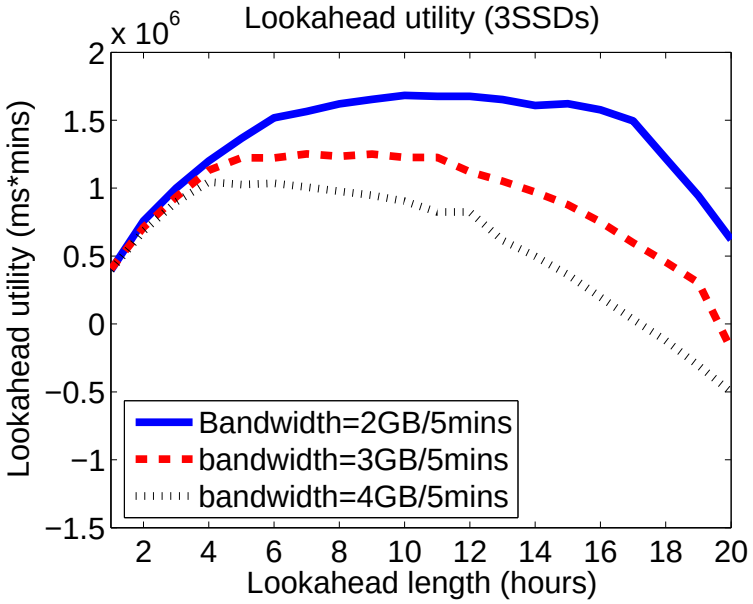

Figure 9: Lookahead utility cost on incremental lookahead length over multiple migration bandwidth for TPCE-SPC1 workload with 3 SSDs

migration on the ongoing workload and the next active workload.

Fig.5 and Fig.6 examines the impacts of lookahead window length on the performance of both first workload and second workload respectively, where lookahead

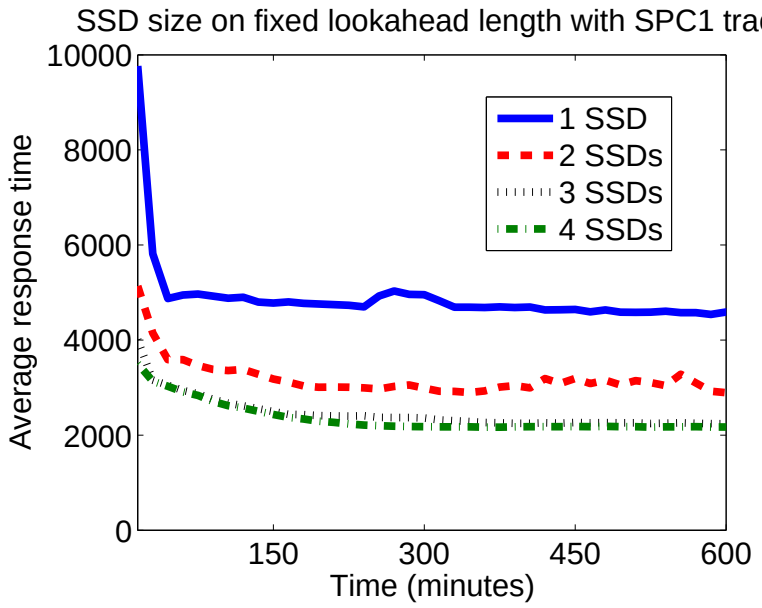

Figure 10: SSD size on response time in fixed 4 hours lookahead length in SPC1 cluster 1 trace with 32 HDDs and 4GB/5mins migration bandwidth allocated

length sets to $2, 4, 6, 8$ hours and the migration bandwidth is set to 2GB/5mins in the TPCE-SPCE workload. Fig.6 shows the impacts of lookahead length on the second workload $w2$. The peak response time of workload $w2$ experiences a continuous dropping as we increase lookahead window length from 2 hour to 8 hours. The peak response time when employing 8 hours lookahead length is 1/3 of the peak response time with 2 hours lookahead length. This is mainly because larger lookahead length enables more hot data extents to be migrated into SSD, given the fixed bandwidth allocated for migration purpose. The more data extents migrated into SSD, the higher reduction in response time is achieved through access disparity between HDD and SSD. Also larger lookahead length enables the migration process to arrive the minimum stable response time level much earlier. This confirms that lookahead window length is a critical knob to control the effectiveness of lookahead migration. Fig.5 shows the impacts of lookahead length on the first workload. It is notable that lookahead migration with window length within the range of ($2 \leq LookaheadLength \leq 8$) introduces almost negligible influence on the ongoing workload $w1$.

Fig.7 and Fig.8 present the results if we continue to increase lookahead length to be over 8 hours. Fig.8 shows that the peak response time of $w2$ is reduced at a much smaller rate. From 10 hours to 16 hours lookahead length, the peak response time of $w2$ is only reduced by about 8%. The reduction on the response time is significantly fading as we continue to increase lookahead length. In other words, the migration effect is saturated when lookhead length is larger than 10 hours. This is because the number of hot extents that are visited in high frequency and thus contribute noteworthy influences on the response time is limited in quantity, if these extents are all migrated into SSD during particular lookahead length, then further increment on lookahead length produces no apparent impacts on reducing response time.

On the other hand, Fig.7 shows that as the lookahead length increases from 12 hours upward the response time of the ongoing workload $w1$ also increases. This is because, the longer the lookahead length is, the more extents of $w1$ are swapped out to accommodate the $w2$ extents. Although the $w1$ extents is swapped in the order from cold extents to hot extents, excessive growth of lookahead length may pose the danger that certain useful hot extents of $w1$ are swapped out, leading to increased response time of workload $w1$.

2.4.3 Optimal Lookahead Length Computation

Fig.9 shows the evolvement of lookahead utility as the lookahead length increments in different bandwidth settings. From the result, we can see that as the lookahead length increments, lookahead utility increases first and then starts to drop at particular lookahead length. For example, when bandwidth is 4GB per 5mins, lookahead utility reaches maximum value at 4 hours lookahead length. This verifies the effectiveness of the lookahead utility model. Also because of larger bandwidth enables the faster migration of hot extents, thus larger bandwidth reaches the maximum lookahead utility values at a faster speed.

Table 1: Optimal Lookahead Length

Bandwidth	Greedy	Formal
2GB/5mins	10 h	10.11 h
3GB/5mins	7 h	7.24 h
4GB/5mins	4 h	4.08 h

Table 1 compares the optimal lookahead migration window length experimented through the use of a greedy algorithm with the optimal lookahead window size computed using our formal model for adaptive lookahead migration, under different bandwidth scenarios with 3 SSDs using the same workload as previous experiments. The results show that our formal model for computing optimal lookahead window size is able to derive the optimal lookahead length quickly and precisely without resorting to a greedy approach. .

2.4.4 SSD size on Fixed Lookahead Length

Fig.10 shows the response time reduction process when we use SSD size ranging from 1 to 4 SSDs. As the SSD size increases, the converged response time is further reduced. The larger the SSD size is, the lower the peak response time will be. As the size of SSD tier increases, more and more hot extents are migrated into the SSDs. Given that the majority of the hot extents are migrated into the SSDs, further increasing the SSD size does not help to further reduce the response time. This can be exemplified by the overlap of the dotted line representing the 3 SSD scenario and the dash-dotted line representing the 4 SSDs scenario.

2.5 Related Work

In storage systems, data migration [83, 49, 139, 19, 112] has been employed to achieve load balancing, system scalability, reliability or a myriad of other objectives. Placing data in different tiers and hence moving data plays an important role in tuning the system resource utilization and guaranteeing quality of service [74, 44]. For example, AutoRAID [127] divides its block storage into a high frequency access layer and a low

frequency access layer. As the block access frequency changes, the system automatically migrates frequently accessed blocks to higher layer and moves less frequently accessed blocks to the lower layer.

Recently, sereral migration techniques have been proposed as a part of the feedback control loop [26]. [25] propose a storage system which evaluates the component's ability in supporting ongoing workload, and selects the candidates to migrate, aiming at satisfying the workload requirements. However, the migration scheme they proposed lacks of consideration on IO bandwidth, SSD size, and migration deadline and other constraints. [83] studies how to use a control-theoretical approach to adjust the migration speed such that the latency of the foreground workload is guaranteed.

2.6 Conclusion

In this chapter we have developed an adaptive lookahead migration scheme for efficiently integrating SSDs into the multi-tier storage systems through deadline aware data migration. Our lookahead migration scheme takes into account several factors in determining the optimal lookahead window size, including the heat information of data extents from the IO profiles, the migration deadline, the migration bandwidth, and the tradeoff between the gain in response time reduction of the workload being migrated to the fast storage tier and the performance degradation of the ongoing workload. The experimental study demonstrates that the adaptive lookahead migration scheme not only enhances the overall storage system performance but also provides significantly better IO performance as compared to the basic migration model and constant lookahead migration scheme.

CHAPTER III

SERVICE MIGRATION IN DECENTRALIZED SYSTEMS

3.1 Introduction

As the cost of the mobile devices and its accessories continue to decrease, there is a growing demand for high performance location based service architecture, aiming at providing scalable and reliable location based information delivery in large scale pervasive computing environments. In contrast to centralized client-server architecture, decentralized management and provision of location based services have gained lot of attentions in the recent years due to its low cost in ownership management and its inherent scalability and self-configurability.

Most of the research and development works in decentralized service computing systems have been focused on unstructured overlay network computing, exemplified by Skype [14], Gnutella [64, 130] and BitTorrent [15] , and structured overlay network systems [118, 101, 108, 140, 16]. Measurements [110, 68] performed on deployed overlay networks show that node characteristics such as availability, capacity and connectivity, present highly skewed distribution [110] and such inherent dynamics creates significant variations, even failures, on the services provided by the overlay systems. For example, a sudden node failure that causes the service interruption may lead the system to exhibit dramatic changes in service latency or return inconsistent results. Furthermore, increasing population size of mobile users and diversity of location-based services available to mobile users have displayed rapidly changing user interests and behavior patterns as they move on the road, which creates moving hot spots of service requests and dynamically changing workloads. Thus an important technical challenge for scaling location service network is to develop a middleware

architecture that is both scalable and reliable, on top of a regulated overlay network with node dynamics and node heterogeneity, for large scale location based information delivery and dissemination. By scalable, we mean that the location service network should provide effective load balancing scheme to handle the growing number of mobile users and the unexpected growth and movements of hot spots in service demand. By reliable, we mean that the location service network should be resilient in the presence of sudden node failures and network partition failures.

In this chapter we present Reliable GeoGrid, a decentralized and geographical location aware overlay network service architecture for scalable and reliable delivery of location based services (LBSs). The main contributions of this chapter are two fold. First, we describe a distributed replication scheme which enables the reliable location service request processing in an environment of heterogeneous nodes with continuously changing workloads. Our replication framework provides failure resilience to both individual node failures and massive node failures, aiming at keeping the service consistently accessible to users and eliminating the sudden interruption of the ongoing tasks. Second, we present a dynamic replica-based load balancing technique, which utilizes a parameterized utility function to control and scale the system in the presence of varying workload changes by taking into account of several workload relevant factors. Our experimental evaluation demonstrates that Reliable GeoGrid architecture is highly scalable under changing workloads and moving hotspots, and highly reliable in the presence of both individual node failures and massive node failures.

3.2 System Overview

Reliable GeoGrid comprises of a network of computing nodes such as personal computer or servers with heterogeneous capacities. The system consists of four core components: topology management module, routing module, replication module and

load balancing module.

Topology management.

All nodes are represented as points in a two dimensional geographical coordinate space, which bears a one-to-one mapping to the physical coordinate system. At any time instant, the network of N nodes will dynamically partition the entire GeoGrid coordinate space into N disjoint rectangles such that each node manages its own rectangular region within the entire coodinate space based on its geographical information and handles all location service requests mapped to its region based on the geographical information of the requests. Figure 11 shows a two dimensional geographical coordinate space partitioned among 17 GeoGrid nodes (for simplicity, we denote each node and its mangaged region with the same number).

GeoGrid is constructed incrementally. It starts with one node who owns the entire GeoGrid space. As a new node p joins the system, it first obtains its geographical coordinate by using services like GPS (Global Positioning System) and then obtains a list of existing nodes in GeoGrid from a bootstrapping server. Then node p initiates a joining request by contacting an entry node selected randomly from this list. The joining request is routed to node q whose region covers the coordinate of the new node. The region owned by q now is split into two halves, one half owned by q and the other half owned by p. In addition to neighbor list, Reliable GeoGrid node also maintains a replica list to provide recovery capability and a routing list for fast routing in the presence of large network size. Mobile users obtain GeoGrid location service by connecting to a GeoGrid node, either through wireless or wired network connections.

In the first prototype design of Reliable GeoGrid, each node is equipped with the capability for submitting location service requests in the form of *Location Query*, routing and processing location service requests, and delivery of results to the mobile users. For example, a car driver may post a service request "send me the traffic

conditions within 5 miles every 10 minutes in the next 1 hour". We assume that location-dependent information sources, such as traffic monitoring cameras, owners of gas stations, and restaurants, and so forth, are external to the service network of Reliable GeoGrid.

Routing Protocol.

Routing in a GeoGrid network works by following the straight line path through the two dimensional coordinate space from source to destination node. A routing request is forwarded initially from its source initiator node to one of its immediate neighbors, say q, which is the closest to the destination location $(x,\ y)$. If $(x,\ y)$ is covered by the region owned by the chosen routing node q, then the node q will be the owner node of this request. Otherwise, q starts the forwarding process again until the request reaches the node whose region covers $(x,\ y)$. For example, in Figure 11, a routing request is initiated by node 7 for a point covered by node 3 is forwarded through nodes 2,4,6 in order and finally arrives its owner node 3.

The other two components of Reliable GeoGrid are replication module and load balancing module, which use replication to provide reliability and scalability for location service. Due to space constraints, the rest of the chapter focuses on these two components.

Each location service request is issued only once and once it is installed into the system, it is read-only and need to be persistent until it expires. The proposed distributed replication scheme replicates all location requests across multiple selected replica hosts in the network and all the replica hosts get the same copy of the read-only location service request, though each location service is only executed at one executor node at any given time. In Reliable GeoGrid, every location service request has an initiator node, an owner node and an executor node. We call the node that receives the location service requests from the mobile users residing in its region *the initiator node* of the services. Each location service request will be routed to the

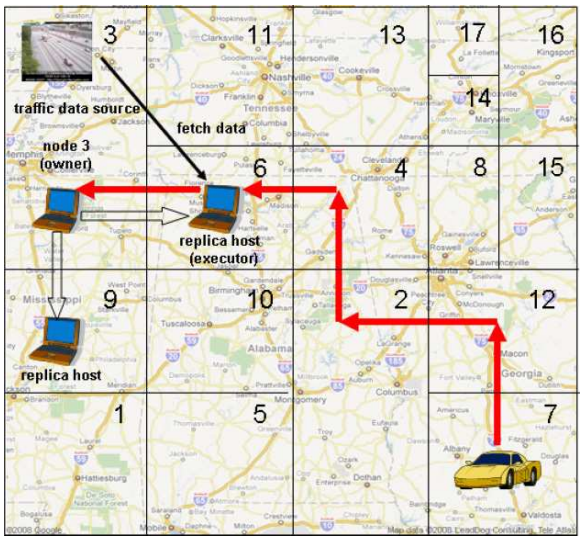

Figure 11: Service Model

destination node whose region covers its geographical coordinate or the coordinate of the center of the spatial area if the request is querying on a spatial region. The destination node is called *owner node* of the location query, which chooses one of its replica nodes to execute the nodes working as *executor node*. When the executor node of a location query fails unexpectedly, one of the replica nodes will be chosen as the new executor node.

3.3 Replication and Replica Management

The GeoGrid replication scheme follows two design principles. First, we want to control the replica management cost by creating and maintaining a constant number of replicas for all services. Second, the replica nodes should be selected from both nearby nodes and remote nodes such that we can take advantage of geographical proximity inherent in the Reliable GeoGrid system to reduce the routing cost involved in recovery and at the same time we can increase the failure resilience of GeoGrid against network partition failures.

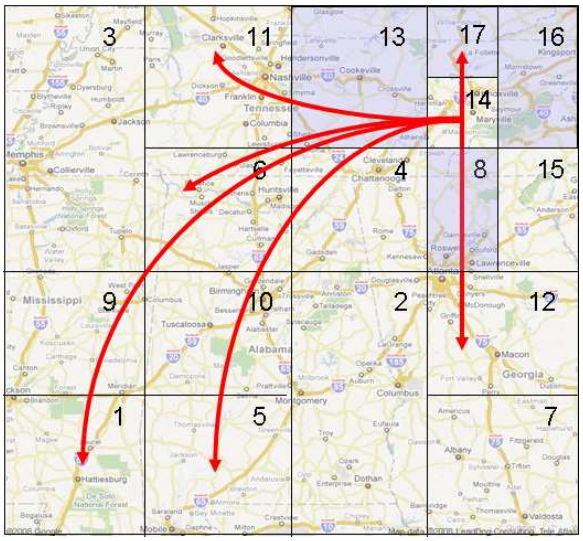

Figure 12: Random replication scheme

3.3.1 Failure Patterns and Risk Analysis

A failure is defined as *an abrupt disconnection* from the Reliable GeoGrid service network without issuing explicit notifications. In practice, such sudden behavior may be caused by computer node crash, network connectivity problems, or improper software termination. Reliable GeorGrid network supports a fail-stop assumption and failures can be captured by prolonged heart-beat messages. By fail-stop assumption we mean that node will stop execution, and lose the contents of volatile storage whenever a failure occurs and node never acts an erroneous action against the system due to a failure [113].

Two types of failures are most common in overlay networks: individual node failures and massive node failures. By individual node failure, we mean that a single node experiences failures independently under fail-stop assumption. Individual node failure may render part of the service interrupted or cause permanent loss of service or service state information or query results, if there is no failure resilience protection employed at the individual node level.

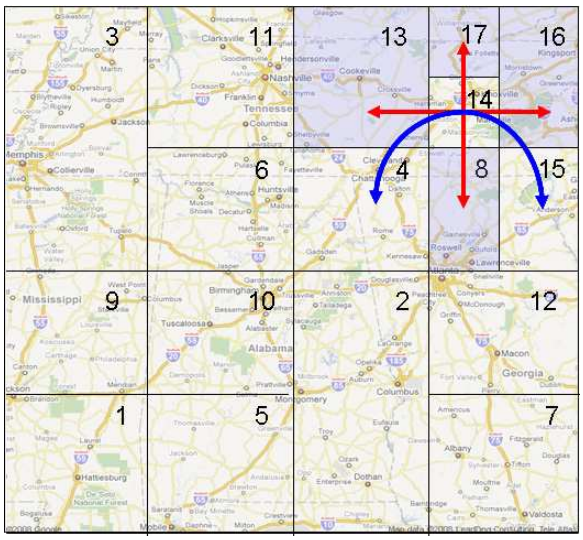

Figure 13: Neighbor replication scheme

By massive node failures we mean that when the underlying IP network partitions, overlay network may partition as well and under such network partitions, the nodes in one partition component are separated from the nodes in another partition component and all the messages across the different network partition components fail to get response and thus create significant delay or critical failures. We argue that a higher level failure resilience mechanism, which can mask the simultaneous failures of multiple nodes, is critical for any long lived large scale systems to maintain service availability.

3.3.2 Baseline Replication Methods

In order to understand the intrinsic factors impacting the design of an effective replication scheme in terms of benefit and cost, we analyze two basic replication methods − neighbor replication scheme and random replication scheme, each of which achieves some degree of failure resilience but suffers from either weak failure resilience or high replica maintenance overhead.

Random Replication Approach.

Random replication is a widely adopted replication method in distributed systems [85,

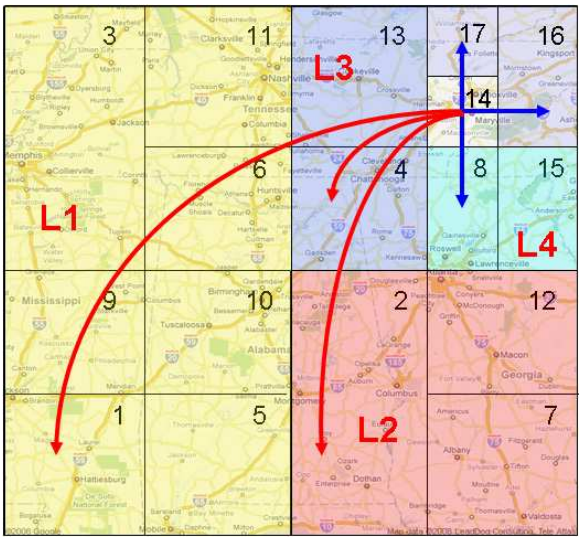

Figure 14: Neighbor-shortcut replication scheme

63]. Given an owner node and a replication factor rf, a random replication scheme will randomly select rf nodes as its replica hosts using a hashing function H.

Because randomly selected replicas are often far away from the host node, random replication exhibits some natural resilience against both individual node failures and massive node failures. However, this approach incurs high replica maintenance cost. First, the random replica hosts may be far away from the owner node in terms of both network proximity and geographical proximity, thus this approach incurs much higher communication and synchronization overheads. Second, if the replica owner crashes, higher overheads in searching and migration are involved to restore the service in the new owner node.

Neighbor Replication Approach.

This replication scheme places replicas in the direct neighbors of the owner node or multi-hop neighbors if the number of direct neighbors is not sufficient to accomplish the replication requirements.

By choosing replica hosts clustered around the owner node, this scheme greatly

reduces synchronization and search overheads compared with the random replication scheme. However, it suffers from the relatively weak resilience to massive node failures. For example, when network partition occurs, if an executor node and its neighboring replica hosts are within the same network segment, then nodes outside this network segment will have no way to reach the location services hosted by this executor node or the service replicas located around this executor node, leading to the unavailability of the services.

3.3.3 Neighbor and Shortcut Replication Scheme

The design objectives of Reliable GeoGrid replication scheme is to provide durable location query maintenance, offer uninterrupted location query processing and enhance the partition tolerance capability. Directed by these objectives, we exploit a hybrid replica placement scheme by combining replication by *"neighboring nodes"* and replication by *"randomized shortcut nodes"*. The former emphasizes that the replica placement should enable fast replica-based recovery and keep the replica maintenance cost low in the presence of high churn rates [104] and node dynamics. The later

promotes the use of shortcut nodes to reach GeoGrid regions that are far away from the region of the current node, which can greatly strengthen failure resilience against severe network partition failures.

Next we describe our replica placement algorithm that chooses neighboring nodes and shortcut nodes as the replica hoarding destination. Finally, we present how Reliable GeoGrid replication scheme dynamically maintains rf invariant replicas in the presence of node departure or node failures.

Overview of Shortcut.

Similar to a path shorter than usual one in real world, *routing shortcut* trims the routing space and reduce the redundant routing hops through maintaining more routing information such as shortcuts to other other larger regions at each node

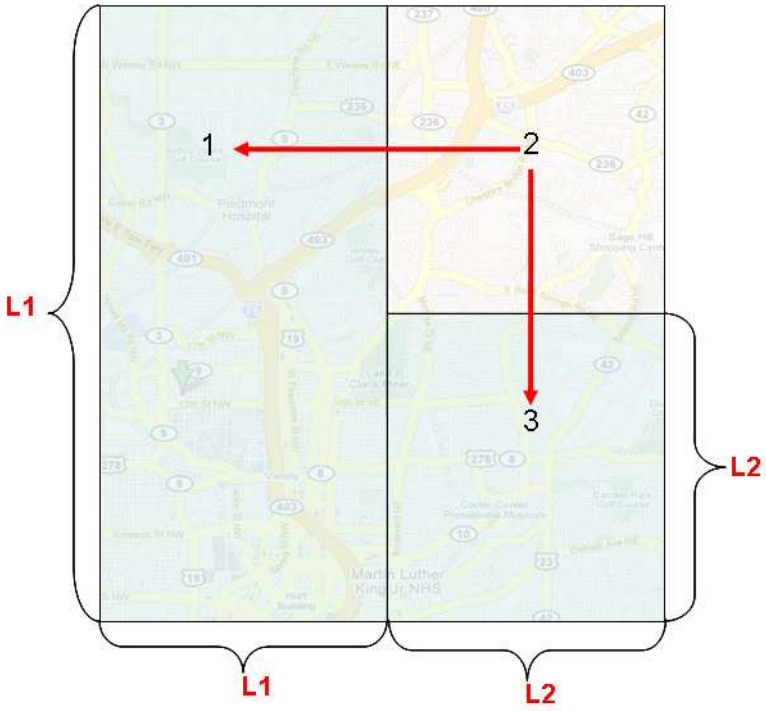

Figure 15: Shortcut list initialization

such that these routing entries can be used as the shortcuts in forwarding routing requests. To build shortcut, in addition to the rectangular regions owned by each GeoGrid node, the entire geographical space is virtually partitioned into a sequence of larger regions such that each region is half size of the previous region in order and are not overlapping with each other, called *shortcut region*. Thus each node stores the addresses of its immediate neighboring nodes but also addresses of one or more residents, *shortcut nodes* for each shortcut regions.

The shortcuts of a node p is organized into a list $L_p < s_1, s_2, \ldots, s_m >$, denoted by $ShortcutList(p)$. m is the number of shortcuts in the shortcut list of p. Each shortcut s_i points to a node in a geographical partition of $1/2^i$ the size of the geographical plane. There are no overlapping among the partitions pointed to by the shortcuts of p.

In Reliable GeoGrid, nodes may have their shortcut lists in different size. The exact length of the shortcut list L_p for a node p is determined by the relative size of

the region R owned by p. When the region R is $1/2^m$ of the size of the geographical plane, the length of the shortcut list L_p is m. This allows the shortcut list of p to cover the entire geographical plane by the shortcuts of p according to the following equation: $\sum_{i=1}^{m} 1/2^i + 1/2^m = 1$.

Based on this analysis, we can estimate the average length of the shortcut list maintained by each node. The size of a region in a GeoGrid of N regions is $\frac{1}{N}$ of the geographical plane, assuming a uniform region size distribution. Thus the length of the shortcut list maintained by each node can be estimated by $O(\log_2 N)$.

As an eample, in Fig.14 node 14 maintains a shortcut pointing to node 1 which is not its direct neighbor. If node 14 is routing a request towards node 9, it can forward this request to node 1 directly which then forwards to node 9. Such routing path effectively trims the half search space, compared with the normal routing path passing node $8, 4, 6, 10$ to reach node 9.

Periodic Refinement of Shortcuts.

To improve the utilization of shortcut, we develop a periodic shortcut refinement algorithm, which is used as an integral part of the shortcut maintenance process. This shortcut refinement mechanism enables the system to balance the distribution of shortcut routing jobs across all nodes in the network, and increase the probability of those nodes with higher capacities and lower workload to serve as shortcuts while preserving the locality of the shortcuts. Concretely, the shortcut refinement algorithm proceeds in two consecutive steps: (1) shortcut selection, which selects a shortcut from the current shortcut list of a node to be replaced; and (2) shortcut replacement, which find a new node nearby using a random walk as the shortcut replacement.

In the shortcut selection step, each node running Reliable GeoGrid middleware will periodically choose a shortcut from its shortcut list as a candidate to be replaced. The actual replacement decision will be made by applying the randomization algorithm. Given a node p with the shortcut list L_p, for each shortcut entry $s_i \in L_p$, p calculates

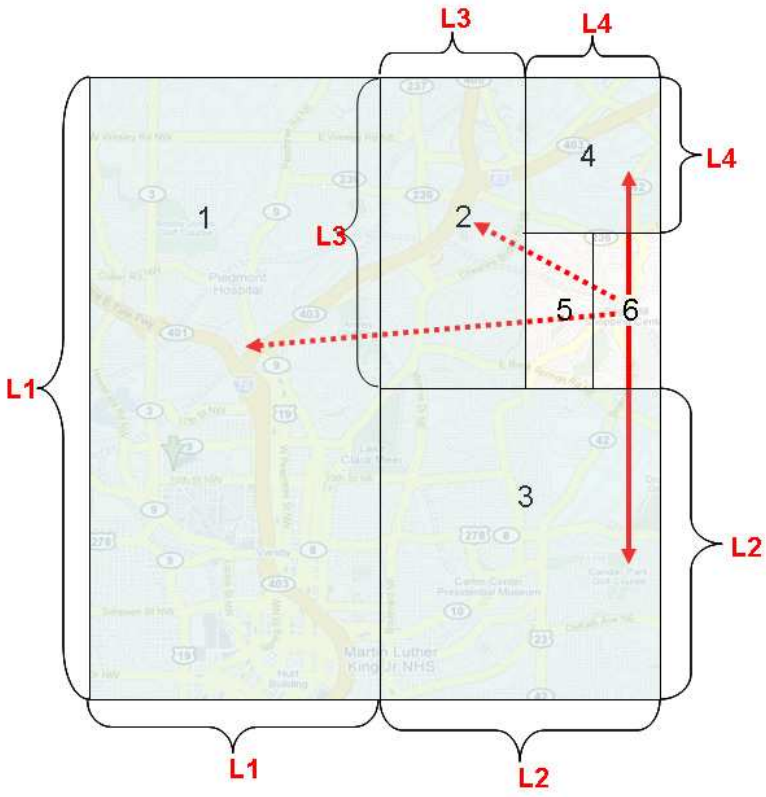

Figure 16: Inherit shortcuts in splitting region

its geographical distance to s_i, denoted by $d(p, s_i)$. A shortcut s_k is chosen with the probability $\frac{d(p,s_k)}{\sum_{s_i \in L_p} d(p,s_i)}$. This probability preserves the approximate locality of the shortcut being replaced. Now node p will make a random decision on whether a chosen shortcut neighbor node s_k should be replaced. The capacity of s_k is compared to the capacity of p weighted by a uniform random number Unif(0, 1). The capacity of a node is computed based on the current workload of the node and its actual capacity when entering the GeoGrid network. Concretely, if $s_k.capacity \leq$ Unif(0,1) $\times$ $p.capacity$, s_k will be replaced. Such a replacement algorithm tends to keep the shortcut to those neighbors that have more capacity, and have higher probability to replace the shortcut node with less capacity.

In the shortcut replacement step, we develop a capacity-aware algorithm for replacing the selected shortcut. We use the node capacity as a criterion when deciding

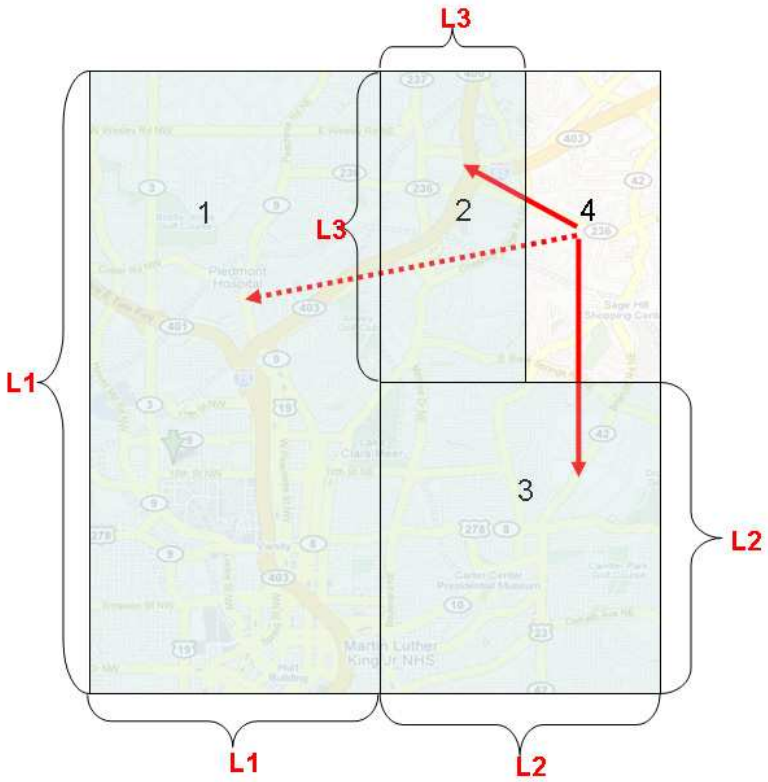

Figure 17: Maintain obsolete neighbor link in joining the system

how and which node should be used to replace the shortcut chosen in Step 1. Intuitively, a node with more capacity and similar locality should be used more frequently as shortcut and handle more routing workload. Concretely, each shortcut selected to be replaced in Step 1 is replaced by a randomly selected node in the same shortcut subtree (the same larger geographical region). The replacement shortcut is chosen by a selective random walk. The random walk is initiated from the node chosen to be replaced. If this node is no longer in the GeoGrid network, its coordinate will be used to locate the new owner node who takes over its region. The random walk is tagged with the information of the node p that initiates the shortcut replacement algorithm. At each step of the random walk, a node q receives the random walk query message from a node p. It will first generate a random number Unif(0,1) and compare $q.capacity$ to $\text{Unif}(0,1) \times p.capacity$. The random walk stops when q has more capacity than p. If the stop condition is not satisfied, node q will select one

of its direct neighbors, which will initiate the next hop of random walk query. A neighbor node n_k is chosen with probability $\frac{n_k.capacity}{\sum_{n_i \in n.neighbors} n_i.capacity}$ $(1 \leq k \leq 4)$. The algorithm in Step 2 ensures that the random walk is directed to and terminated at the node with more capacity than the one being replaced while keeping the proximity constraint (i.e., the region owned by this chosen node is close to the geographical region of the node being replaced).

Replication Factor rf. Given a location query request LQ, let node p be either the owner node or executor node of LQ. LQ will be replicated at the following set of nodes:

$$ReplicationList(p, lq) = [(p_1), (p_2), \ldots, (p_{rf})], \text{ where}$$
$$\bigwedge_{k=1}^{rf} p_k \subset \{\text{NeighborList(p)} \cup \text{ShortcutList(p)}\}$$

This set is called the *Replication List* of node p, denoted by $ReplicationList(p)$. As a location query lq is issued and received by its owner node, it is replicated to *ReplicationList* of the owner node p. The length of the replication list is defined by the *replication factor* rf, which is a tunable system supplied parameter, set in the system initialization time and continually tuned according to failure rate, throughput of the system, and the latency of the messages. Setting rf to a large value may cause the system to pay higher replica maintenance cost for fault tolerance and such cost can be further aggravated due to high churn rate of the overlay network or fast movement of hotspots in terms of request patterns of mobiles. Another important design consideration is to keep the ratio of neighbor replica and shortcut replica to be relatively proportional and constant for each node p, because shortcut nodes are usually further away from node p and it is important to keep sufficient number of neighboring nodes as the replica hosts. In the situation where the rf value is large, and combining both the neighbor list and the shortcut list of a node p is insufficient to fulfill the replication requirement, i.e., size(NeighborList(p)) + size(ShortcutList(p))

$< rf$, we will extend $NeighborList$ to the multi-hop neighbor list which maintains the i-hop neighbor list of a node p for $i = 1, \ldots, k$. As shown in Figure14, with $rf = 6$ and the minimum neighbor replica to be 50%, node 14 selects its 3 direct neighbors: node 8,16,17, and its 3 shortcut nodes: node 1,2, 4 to compose its replica list.

The dynamic replica management module maintains the rf replicas for each node in the network when node leaves or joins the network by monitoring the rf number of replica hosts in the *ReplicationList* with the help of lower level Reliable GeoGrid operations such as periodic heartbeat messages.

3.4 Load Balancing Through Replication

An important challenge in scaling pervasive location service is the system-level capability in handling continuously changing hot spots in terms of service demands and access patterns of mobile users.

In Reliable GeoGrid, we designed a dynamic utility-aware, replica-based load balancing scheme, which takes into account load per node, cache factor, and network proximity factor to exploit the service processing capabilities of replicas. Specifically, upon the detection of overload at a node, the load balance algorithm will be turned on. For each newly arrived LQ, the selection of executor node for a LQ takes into account three factors that have critical impacts on load balance and system resource utilization. The first factor is the load per node, namely how much load does a replica host currently have. The second factor is the cache affinity factor, which states whether the data items interested by the location query is in the cache of the replica host. The third factor is the network proximity of the replica host to the remote data source that provides the data items of the given query. By taking into account the runtime load level of node p and its replica hosts, we can avoid the risk of offloading the LQ from the node p to another heavily loaded replica node and create unwanted query drops, at the same time we increase the probability of assigning the

LQ to a node that has more resources and yet less loaded. By taking into account the cache affinity factor, the node with required cache items will be ranked higher in its load balancing utility value, thus we avoid repeated and blindly data fetching from the remote data source and effectively reduce the query processing overheads of the system. By considering the network proximity between the replica host node and the remote data source being queried, better system utilization is achieved. We design a weighted utility function based on the above three factors, and compute a utility value for each candidate node and select the node with the largest utility value as the query executor for load balance. Formally, the utility function computes the utility value as follows, where p denote a node in the replication list, and lq denotes the location query to beselecting executor.

The notion of the load on a node p is formalized as the total query service processing costs for all the active location queries allocated to the node p divided by its capacity, which is denoted as "workload index". Let $p.propos.workloadIndex$ denotes the workload index of p and $QueryDropThreshold$ denote the query drop threshold. We define the concept of balance as $balance$=$p.propos.workloadIndex - tresh * QueryDropThreshold$, where $tresh$ controls the weight of query drop threshold with respect to per node workload index. We measure the load level of node p in terms of whether node p accepts one more location query, denoted by $p.load\_level$,

as follows:

$$p.load\_level = \begin{cases} 1 & \text{if } balance \leq 0 \\ 1 - \frac{p.propos.workloadIndex}{\text{QueryDropThreshold}} & \text{if } balance \geq 0 \end{cases} \tag{12}$$

cache_factor measures the possbility that the data item being queried by the query is contained in the candidate node's cache. It is defined as follows:

$$cache\_factor = \left\{ \begin{array}{ll} 1 & \text{if } lq.data\_item \text{ is in } p.propos.cache \\ 0 & \text{otherwise} \end{array} \right\} \quad (13)$$

proximity_factor is a measure of the network proximity of the candidate node p to the data source node s. $proximity\_factor$ is defined as follows:

$proximity\_factor = \frac{1}{distance(candidate\_node, data\_source)}$

Let $Utility(q)$ denote the utility function, which computes a utility value for selecting node q as the query executor for the purpose of load balancing. The utility function is defined as follows:

$Utility(q, cq) = load\_level * (cache\_factor + \alpha * proximity\_factor(candidate\_node, data\_source))$

Note that the sum of $cache\_factor$ and weighted $proximity\_factor$ is multiplied with node load factor $load\_level$. This gives more weight to the node load factor. For instance, a node which is close to the data source but is heavily overloaded will not be selected as the query executor. α is system tunable parameter to adjust the importance weight of the source distance factor on the cache factor.

In our first prototype and our experimental evaluation, we set α to be 0.4

3.5 Experimental Results

This section reports our experimental results for Reliable GeoGrid service network by simulating a geographical region of 64 miles × 64 miles. The population of end users in this region ranges from 1×10^3 to 1.6×10^4. For each population, we simulated 100 randomly generated Reliable GeoGrid networks. Each end user connects into the Reliable GeoGrid system through a dedicated proxy node. The capacities of those proxies follow a skewed distribution using a measurement study documented in [109].

We report two sets of experiments that evaluate the effectiveness of Reliable GeoGrid approach to scaling location service networks. We first study the fault tolerance of our replication scheme against individual node failures and massive node failures.

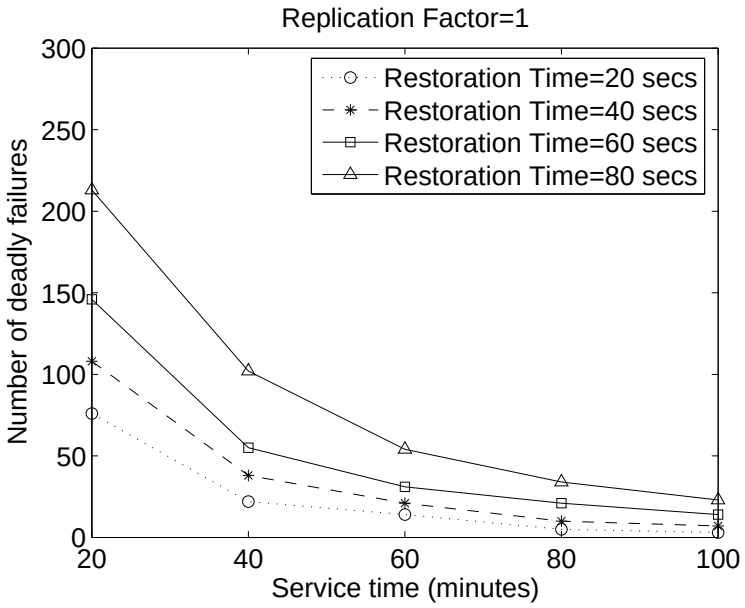

Figure 18: Deadly failures,rf=1,system size 4000 nodes

Then we evaluate the effectiveness of our utility-aware, replica-based load balancing scheme.

3.5.1 Failure Resilience

There are two common types of node failures: individual node failure, and massive node failure. In the case of individual node failure, without replication, the LQs hosted by the failed node p will be lost, though the geographical region, for which the failed node is responsible, will be taken over by one of the neighbor nodes upon detecting the departure of node p. However, with our replication scheme, such individual node failure will ensure no interruption of the system operation at all since all LQs hosted by the failed node will be recovered by one of its rf replica nodes, assuming that not all rf replicas failed together. Otherwise, a critical failure will occur.

Fig. 18 and Fig. 19 plot the total number of critical failures captured during this experiment under different settings of mean service time (st), restoration time (rt) and replication factors (rf). We observe that the neighbor and shortcut replication scheme can help the system to significantly reduce the occurrence of critical failures.

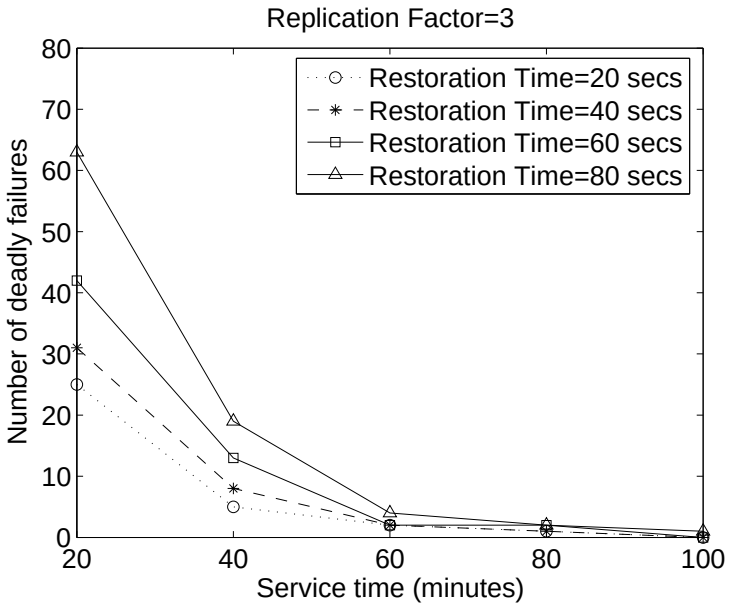

Figure 19: Deadly failures, rf=3, system size 4000 nodes

With larger replication factor, smaller restoration time and longer service time, we can achieve better reliability and incur less number of critical failures. As defined earlier, critical failure occurs when all replica hosts of a LQ fail within the restoration time interval Δt_r. Furthermore, the simulation models the worst scenario where nodes leave the system upon failure but no new node enters the system, leading to more critical failures. In Fig. 19 a moderate replication factor 3 will reduce the number of critical failures to almost zero for system with service time more than 40 mins. This shows that the proposed replication scheme can significantly reduce the number of critical failures and achieves reasonable reliability through placing moderate number of replicas.

Everytime the system migrates a replica from one node to another node data movement overheads occur, we denote it as individual replica migration overheads, which is defined as $dataSize * communicationLatency$. Thus the system replica maintenance overheads is defined as the sum of all the individual replica migration overheads. Fig.21 shows the comparison between achieved reliability and replication maintenance overheads. As shown by the dotted line with hollow circle marker,

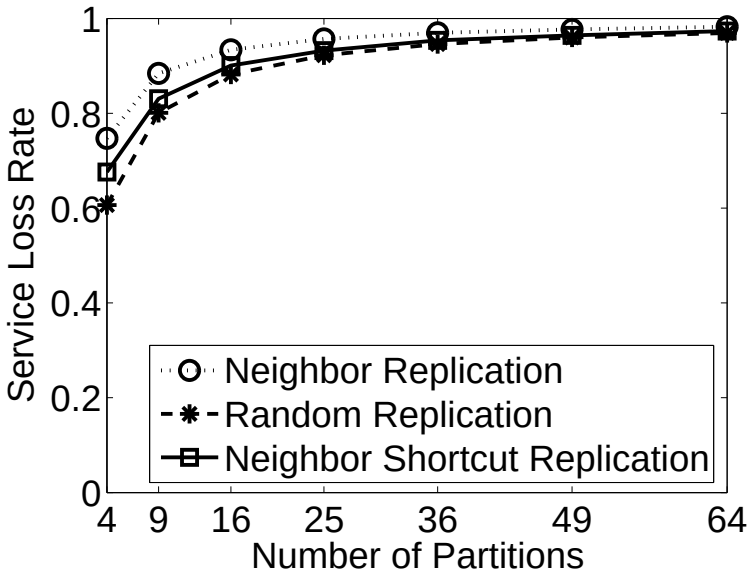

Figure 20: Service Loss Rate in massive node failures rf=2 system size 10000 nodes

dashed line with solid circle marker and solid line with square marker, higher reliability is achieved as rf increases. The dotted line with triangle marker shows how the replication maintenance overheads increases as the replication factor increments. For a system with 4000 nodes with service time around 20 mins (high node dynamics), replication with rf=4 introduces relatively low overheads while achieving good reliability.

Fig. 20 and Fig. 22 compare the failure resilience of the three replication schemes discussed earlier in the presence of network partition failures.

We examine the cases with the number of network partitions ranging from 4 to 64 for a 10000 nodes network with 5000 random LQs. We use the *service loss rate* to measure the failure resilience in such scenario, which is defined as the percentage of unsuccessful LQs. Overall, we observe that as the number of overlay network partitions increases, the service loss rate increases in all three replication schemes, and the loss rate of random replication approach and the lost rate of the neighbor-shortcut replication approach start to converge. For an overlay network with 36 network partitions or higher, the random replication approach performs only slightly

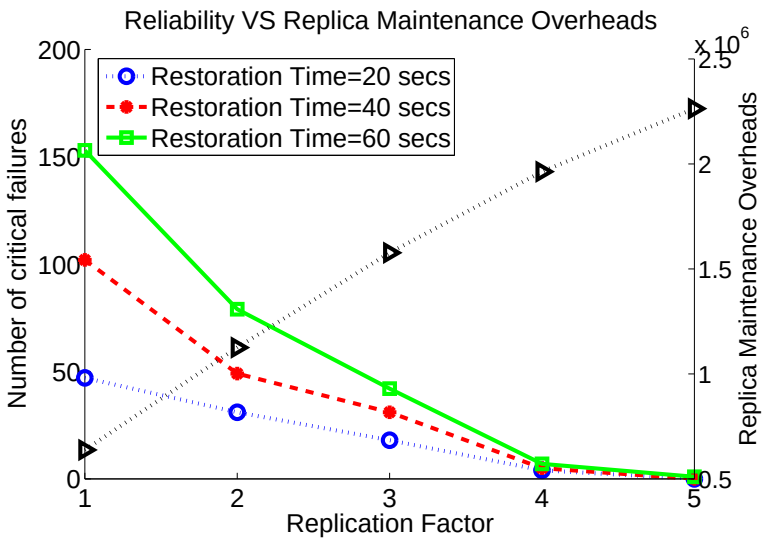

Figure 21: Reliability VS Replica maintenance overheads (system size 4000 nodes service time 20 mins)

better than the neighbor-shortcut replication approach in terms of service lost rate.

We also observe that the higher the replication factor rf is, the more effective our replication scheme performs in terms of the service loss rate reduction. Fig.23 compares the replication maintenance overheads among these three replication schemes. The result confirms that random replication incurs the most maintenance overheads and neighbor replication scheme introduces the least amount of maintenance traffic while neighbor and shortcut replication scheme is in the middle. Combining the failure resilience towards both individual node failure and network partition failure and the moderate maintenance overheads, we can conclude that neighbor and shortcut replication scheme takes the advantages of the two replication schemes while avoiding their weakness and can achieve reasonable good reliability through placing moderate number of replicas.

3.5.2 Evaluation of Load Balance Scheme

To evaluate the effectiveness of the proposed load balance scheme in dealing with continuously changing workload in pervasive computing environment, we built a discrete

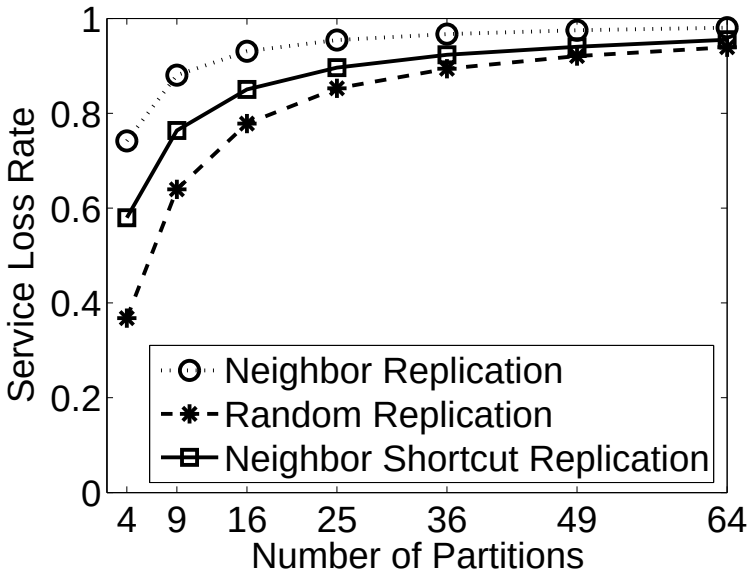

Figure 22: Service Loss Rate in Network Partition Failures (rf=4,System Size=10000 nodes)

event simulator that models the allocation of the location query events to nodes.

The workload is a mixture of regular location query service requests and moving hot spot query requests.

Fig. 24 examines load balance scheme performed on regular uniform workload distribution (without hot spot) and we set rf as 4 and epoch length as 20 seconds with a total of 50000 regular location query requests. We observe that as the system size increases, the standard deviation values of the workload index decreases and the load balance scheme reduces the standard deviation of the workload index to almost 10 percent of the case without load balance.

For a system without load balance scheme equipped, the occurence of hot spot may introduce longer service latency because of the longer waiting queues. To study how well the load balance scheme helps to reduce the query latency when using load balance scheme, we define "latency stretch factor" as the ratio between the average latency in the case without load balance scheme activated and the average latency when the replica based load balance scheme is activated. In other words,

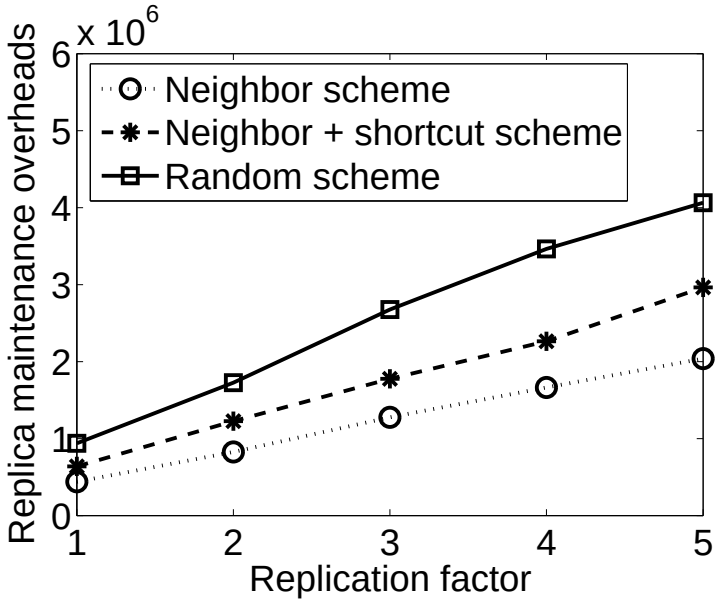

Figure 23: Repliation overhead comparison on replication schemes (System Size=4000 nodes)

higher latency stretch factor indicates higher query latency caused by the hot spot. Fig.25 shows the experimental results in different system size. We can observe that a replication factor 4 for 8000 nodes system and replication factor 5 for 4000 nodes system can eliminate the extra query latency caused by hot spot and reduces the query latency to the same scale as the case without hot spot, indicating by query latency stretch factor equivalent to 1. This shows that the replica based load balance scheme greatly helps the system to reduce the query latency and improve the system performance even when hot spot occurs.

Fig. 26 plots the experimental results on dynamic moving hotspot workload, where 5 hotspots with diameter 10 are generated. The solid line represents the measured results of the "moving hot spot" when load balance scheme with threshold 0.1 is activated and dashed line represents the case with threshold 0.5. The dotted line represents the performance of Reliable GeoGrid system without load balance scheme.

We observe that the standard deviation of Reliable GeoGrid system without load balance presents chaotic features because moving hot spots generate high volume

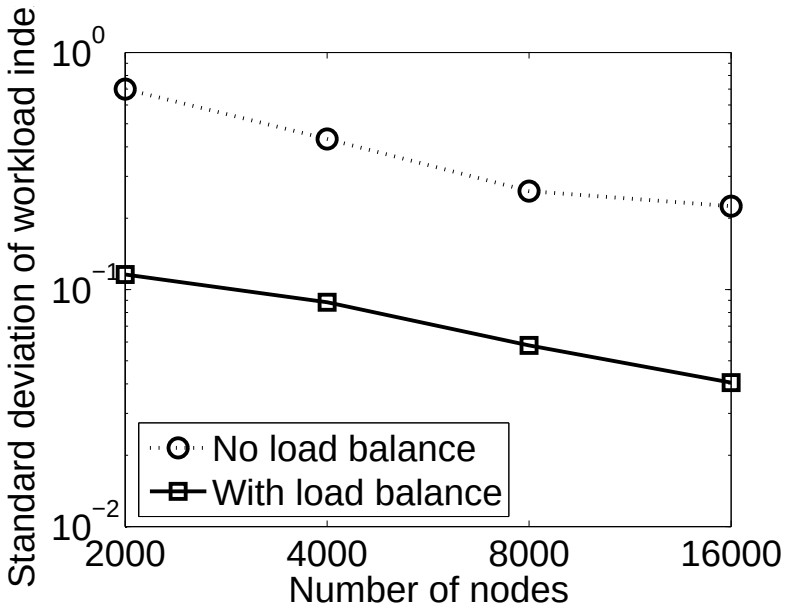

Figure 24: Standard deviation of workload index (no hot spot) on rf=4

of location queries in some areas. In contrast, Reliable GeoGrid, powered with our replication-based, utility-aware load balance scheme, can effectively maintain the stable performance while the system workload experiences chaotic perturbation.

Figure 27 shows that under varying threshold and varying hot spot diameter size from 5 to 25, there is no query dropped. However, in the case without load balance, query drop rate continously increases, up to 2.5% at diameter size of 25. This shows the load balance scheme achieves better overall system resource utilization.

3.6 Related Work

We have presented Reliable GeoGrid for providing reliability and scalability in large scale location service network with inherently heterogeneous and possibly unreliable nodes. Location based service is gaining much popularitiy for its ability to provide users with precise location related service [57, 58, 62]. Compared with a centralized managed geographical location service approach which suffers the long response time delay, expensive service access and vulnerability to sudden surge of hot spots, decentralized overaly network tackles the scalability bottleneck through distributing

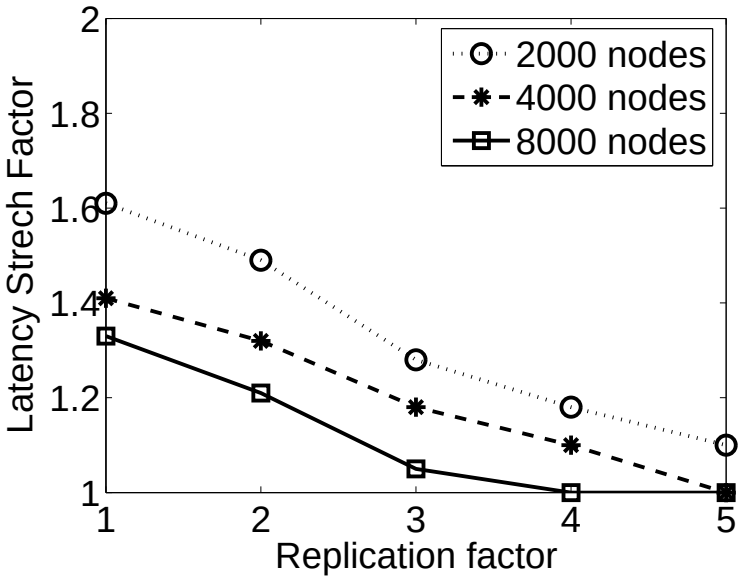

Figure 25: Impact of replication factor (rf) on load balance

the workloads among the large scale participating nodes and eliminates the needs for server side infrastructure support. In addition to location based service, due to its scalabiltiy and flexibility, overlay network attracts intensive research efforts on building scalable applications on top of it. For example, VOIP applications [14, 31], publish-subscribe services [29, 35, 36, 71, 41, 69, 89, 102, 107] and storage system [48, 76]

Also there have been intensive research efforts on serving location based services and applications in mobile environments through an overlay network [124, 128, 53, 106]or a network of caches [115, 39]. Some research works like Siena [35], Rebeca [89],[53],[70],[40] and [77] focus on location based query evaluation with publish subscribe networks. However, most of these research works focus on improving the scalability and proximity awareness of location based services by utilizing its decentralized, self-managing features. Only few research has been focusing on improving the failure resilience of overlay network related location based service. It is widely recognized that achieving reliability is critical for future deployment of P2P technology to provide quality location service for growing number of users.

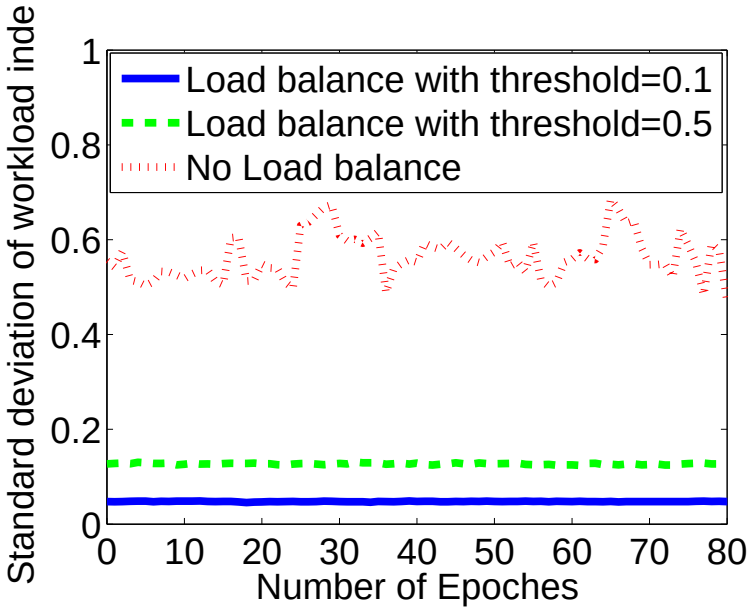

Figure 26: Hot spot diameter on Standard deviation of workload index, rf=4

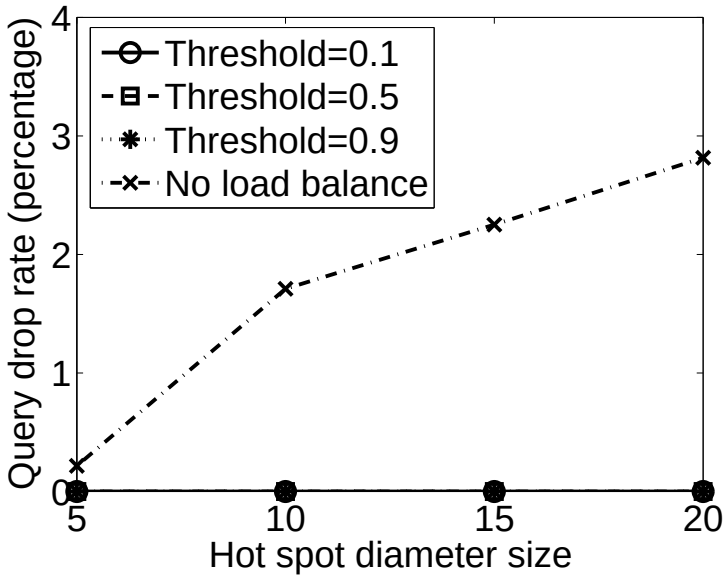

Figure 27: Service drop rate comparison with no load balance, rf=4

Facing the heterogeneous nature of node capacity and availability, fault tolerance in overlay network creates important impacts on the system performance [32, 52]. Replication [22, 23, 66, 133] is a basic approach to achieve failure resilience [113],which has attracted intensive research efforts to apply it in different areas [97, 143, 60]. In some overlay networks, replication is applied as an accessory tool to improve the reliability such as autonomous replication [45, 142, 43], and symmetric replication [63, 90]. Existing research on replication scheme for overlay network can be classified into two categories: random replication and cluster replication. Random replication [85, 63] scheme is widely used in distributed files systems. Cluster replication [46, 61] places the replicas around the owner node. The most representative examples are the neighbor replication in Chord [46, 118]. However, most of these replication research works focus on individual node failures and less research works study the massive node failures: network partition failure [95, 114, 42, 50] .

In practice network partition failure is also one type of failures that happens very often and may create more serious damage on system than individual node failures [123, 28]. Traditionally, network partition was studied in the areas such as file systems [111, 97, 28, 121] and data management [42, 50, 23]. Surprisingly very few work has addressed this type of failures in the context of overlay network. In our work, the replication scheme we proposed can improve failure resilience against both individual node failure and network partition failures. Also Reliable GeoGrid replication scheme adopts the raw replication, which eliminates the overheads and latency to reassembly the fragments compared with the erasure coding scheme [126].

As important as reliability, load balance is also a key objective that distributed system research aims to improve. There have been intensive research works on load balance in distributed systems [27, 55, 59, 119, 122, 100]. Most of existing load balance schemes [99, 103] and [65] focus on random workload reassignment among nodes which incurs high overheads while reduces the burden of some heavily loaded

nodes. Although these solutions can reduce the workload from heavily loaded nodes, it requires to maintain a few centralized directory nodes and each node needs to keep the identification information of all the directory nodes. Furthermore, the selection of the load shedding node fails to consider the communication bandwidth and promixity distance factor which may also lead to high communication overheads.

3.7 Conclusion

We have presented Reliable GeoGrid, a location-based service overlay network for scaling location based services and enhancing reliability of pervasive computing applications. This chapter makes three original contributions. First, we developed a methodical approach to building a reliable and scalable location service network with neighbor-shortcut based replications. Second, we developed a dynamic replica-based load balancing scheme with a utility-aware model, which takes into account node heterogeneity, network proximity, and changing workload at each node to scale the system in the presence of unexpected workload changes and node failures. Third but not the least, our prototype and experimental study demonstrates that the Reliable GeoGrid framework is highly scalable in terms of changing hotspots and highly reliable in the presence of both node failures and network partition failures.

CHAPTER IV

APPLICATION MIGRATION AND VALIDATION

Cloud computing infrastructures such as Amazon EC2 [1] provide elastic, economic, scalable, secure solutions for users. Its pay-as-you-go paradigm attracts many enterprises to build their services or applications on the EC2 platform and many successfully achieve their business objectives, such as SmugMug, Twistage and etc. More and more enterprises embrace Cloud computing by making their plans or are in the process to migrate their services or applications from local data center to the Cloud computing platform like EC2, because this will greatly reduce their infrastructure investments, simplify operations, and obtain better quality of information service.

However, the application migration process from the local data center to the Cloud environment turns out to be quite complicated: error-prone, time-consuming and costly. Even worse, the application may not work correctly after the sophisticated migration process. Existing approaches mainly complete this process in an ad-hoc manual manner and thus the chances of error are very high. Thus how to migrate the applications to the Cloud platform correctly and effectively poses a critical challenge for the research community.

In this chapter, we plan to investigate the causes that make the migration process complicated and error-prone, design the migration approach and algorithms to simplify and facilitate this process and conduct experiments to verify the efficacy of the proposed approach.

In the following sections, we first discuss the potential causes that lead to the complicated and error-prone nature of the migration process. Next we introduce the existing approaches and their limitations. Then we introduce the user study we have

conducted on the migration problem and draw the keys to solve this problem. Next we discuss the CloudMig system, which provides both the configuration validation and installation automation to simplify the migration process.

4.1 Why Migration to Cloud is Complicated and Error-prone

There are some causes that make the migration process to Cloud complicated, error-prone and prohibitively expensive. First, the computing environmental changes render many environment dependent configurations invalid. For example, as the database server is migrated from local data center to the Cloud, the IP address is possibly changed and this inevitably imposes the requirement of updating the IP address in all the components that depend on this database server. The migration process incurs large number of configuration update operations and even a single negligence of a single update may render the whole system out of operation. Second, the deployment of today's enterprise system consists of large number of different components. For example, for load balancing purpose, there may be multiple web servers and application servers in the systems. Thus the dependencies among the many components are rather complicated and can be broken very easily in the migration process. Sorting the dependency out to restore the normal operational status of the applications may take much more time than the migration process itself. Third, there are massive hidden controlling settings which may be broken inadvertently in the migration process. For example, the access controls of different components may be rumpled, which confront the system to the security threats. Lastly, the human operators in the complicated migration process may make many careless errors which are very difficult to identify. Overall, the complicated deployments, the massive dependencies, and the lack of automation make the migration process difficult and error-prone.

4.2 Related Work

Most of the existing migration approaches are either manually based or only limited to certain types of applications. For example, the suggestions recommended by Opencrowd are pretty high level and abstract and lack the concrete assistances to the migration problem [2]. The solution provided by OfficetoCloud is only limited to the type of Microsoft Office products and does not even scratch the surfaces of large application migration [8]. Thus, a thorough and realistic study on the complexity of migrating large scale applications to Cloud is essential to direct the Cloud migration efforts. Furthermore, an automatic and operational approach is highly demanded to simplify and facilitate the migration process.

Nagaraja et al. [91] proposed a testbed for inserting faults to study the error behaviors. In our study, we study the operator errors by migrating real practical applications from local data center to EC2. This forms a solid problem analysis context which motivates the effective solution for the migration problem. Vieira and Madeira [125] proposed to assess recoverability of database management systems through fault emulation and recovery procedures emulation. However, they assumed that human operators had the fault identification capability. In our work, we assume that human operators only have certain error identification capabiltiy but still can not avoid errors. Thus an automated configuration management system is highly demanded. There is already intensive research work on design, evaluation of interactive systems with human operator involved in the field of human computer interaction. For example, Maxion and Reeder in [87] studied the genesis of human operator errors and how to reduce them through fine user interface. However, there is no research work which proposes to build the policy checking system based on continual query to check the configuration data. Our work is the first research effort in exploring the continual query based policy checking system to simplify the application migration process from

local data center to Cloud by fully exploiting the monitoring and checking functionality of continual query [78, 82, 79, 81].

4.3 Migration Operations and Error Model

In this section, we introduce a series of application migration practices we conducted in migrating typical applications from a local data center to EC2 Cloud platform. We first introduce the experimental setup and then we discuss the migration practices on representative applications in details, in particular on the errors made during the migration process. Based on this, we build the migration error model through the categorization of the migration errors.

4.3.1 Experiment Setup

Our experimental testbed involves both a local data center and EC2 Cloud. The local data center in College of Computing, Georgia Institute of Technology, is called "loki", which is a 12-node, 24-core Dell PowerEdge 1850 cluster. Because the majority of today's enterprise infrastructures are not virtualized, the physical to virtual migration (P2V) migration paradigm is the mainstream. In this work, we focus on P2V migration.

We deliberately select representative applications as migration subject. These applications are first deployed in the local data center and then operators are instructed to migrate the applications from local data center to the Cloud. With the experimental setup across the local data center and Amazon EC2 platform, we are able to deploy moderate enterprise scale of applications for migration from a real local data center to the real Cloud platform and test the hyperthesis under the setting of real workload, real massive systems, and real powerful Cloud.

We select two types of applications in the migration experiments: Hadoop and RUBiS. These applications represent typical types of applications used in today's enterprise computing. We select these applications mainly from the consideration

dimensions of service type, architecture design and migration content.

1. Hadoop [4], as a powerful distributed computing paradigm, has been increasingly attractive to many enterprises to analyze large scale data generated daily, such as Facebook, Yahoo, etc. Many enterprises utilize Hadoop as a key component to achieve data intelligence. Because of the distributed nature, the more nodes participating in the computation, the more computation power is obtained in running Hadoop. Thus, when the computation resources are limited at local site, enterprises tend to migrate their data intelligence computation applications to Cloud to scale out the computation. From the aspect of service functionality, Hadoop is a very typical representation of the data intensive computation applications and thus the migration study on Hadoop provides us good referential value on data intensive application migration behaviors.

   Hadoop consists of two subsystems, map-reduce computation subsystem and Hadoop Distributed File System (HDFS), and thus migrating Hadoop from local data center to the Cloud includes both computation migration, file system migration or data migration. Thus it is a good example of composite migration. From the angle of architecture design, Hadoop adopts the typical master-slave structure in its two layers of subsystems. Namely, in map-reduce layer, a job tracker manages multiple task trackers and in the HDFS layer, a NameNode manages multiple DataNodes. Thus the dependency relationships among multiple system components forms a typical tree structure and the migration study on Hadoop reveals the major difficulties or pitfalls in migrating applications with tree-style dependency relationships.

   In our P2V experiment setup, we deploy a 4-node physical Hadoop cluster, and designate one physical node to work as NameNode in HDFS or job tracker in map-reduce and four physical nodes as DataNode in HDFS or task tracker

in map-reduce (the NameNode or job tracker also hosts a DataNode or task tracker). The Hadoop version we are using is Hadoop-0.20.2. The migration job is to migrate such source Hadoop cluster to the EC2 platform into a virtual cluster with 4 virtual nodes.

2. RUBiS [9]is an emulation of multi-tiered Internet services. We select RUBiS as the represntative case of large scale enterprise services. To achieve the scalability, enterprises often adopt the muti-tiered service architecture. Multiple servers are used for receiving Web requests, managing business logic, and storing and managing data: Web tier, application tier, and database tier. Depending on the workload, the number of servers in certain tier can be added or reduced by adding more servers or removing some existing servers. Concretely Apache HTTP server, Tomcat application server and MYSQL database work as the Web tier, application tier and database tier respectively.

   We select RUBiS benchmark as a migration subject by considering the following factors. First, Internet service is a very basic and prevalent application type in daily life. E-commerce enterprises such as EBay, usually adopts multi-tiered architecture as emulated by RUBiS to deploy their services and this renders RUBiS to be a representative case of Internet service architecture migration. Second, the dependency relationship among the tiers of multi-tiered services is a graph and this makes it distinguished in studying the dependency relationship perservation during the migration proces. Third, the migration content of this type of application involves reallocation of application, logic and data and thus its migration provides a good sample study on richful content migration. In the P2V experiment setup, one machine installs the Apache HTTPD server as the first tier, and two machines install the Tomcat application server as the second tier, and two machines install the MYSQL database as the third tier.

In the following subsections, we introduce two case migration studies we conducted: Hadoop migration and RUBiS migration. The errors are mainly configuration errors and installations errors. The configuration errors are our foucs because they are the typical operator errors and difficult to identify and correct. Installation errors can be corrected or elimiated by more organized installation steps or automatic installation tools or more detailed installation instructions. We first discuss the typical configuration errors in each migration case study based on our categorization scheme which classifies the configuration errors into seven categories. Next we introduce the common installation errors across both case studies. For each category of the error, we illustrate with a few typical example errors in our experiemnts. However, these examples are just a subset of the errors present in experiments. Finally we present the statistics results on the error distributions in each case study and analyze the major erorrs which lays a solid foundation for the design of the configuration management system.

4.3.2 Hadoop Migration Study

In this experiment, we migrate the source Hadoop application from the local data center to EC2 platform. In this section, we discuss the typical configuration errors present in this process.

Dependency Preservation This is the most common error present in our experiments. Such pitfall is very easy to make and very difficult to discover and may lead to disaster results. According to the degree of severe impacts of this type of error on the deployment and migration, it can be further classified into four levels of errors.

The first level of "dependency preservation" error is generated when the migration administrator does not realize the necessity of dependency preservation checking. Even the dependency information presents explicitly, lacking of initiative to review

the components dependency may lead to stale dependency information. For example, in our experiments, if the migration operator forgets to update the dependency information among the nodes in the Hadoop application, then the DataNodes (or task tracker) will still initiate the connection with the old NameNode (or job tracker). This directly renders the system unoperational.

The second level of this type of error is making typos in the dependency files. For example, the typo hidden in the host name or IP address renders some DataNodes to be unable to locate the NameNodes.

The third level of this error type is incomplete dependency constraints update. For example, one operator only updated the configuration files named "masters", and "slaves" which record the NameNode and list of DataNodes respectively. However, Hadoop dependency information is also located in some other configuration files such as "fs.default.name" in "core-site.xml" and "mapred.job.tracker" in mapred-site.xml. Thus Hadoop was still not able to boot with the new NameNode. This is a typical pitfall in migration, and is also difficult to detect by the operator because the operator may think that the whole dependency is updated and may spend intense efforts in locating faults in other locales.

The fourth level of this type of error is insufficient number of updated machines. That is, although the operator realizes the necessity to update the dependency constraints and also identifies all the contraints locations, but is not able to update all the machines in the system which is involved in the dependency constraints. For example, in Hadoop migration, if not all the DataNodes update their dependency constraints, the system can not run with the participation of all the nodes.

Network Connectivity Bearing the distributed computing nature, Hadoop involves intensive communication across nodes in that the NameNode keeps communication with DataNodes and job tracker communicates with task tracker all the time. Thus for such system to work correctly, inter-connectivities among nodes become an

indispensible prerequisite condition. In our experiments, operators showed two types of network connectivity configuration errors after migrating Hadoop from local data center to EC2 in theP2V migraton paradigm.

The first type of such error is that some operators did not set the network to enable all the machines to be able to reach each other over the network. For example, some operators forgot to update the file "/etc/hosts" and led to IP resolvement problems. The second type of such error is local DNS resolution error. For example, some operators did not set the local DNS resolution correctly, which led to that only the DataNodes residing in the same host as master node was booted after the migration.

Platform Difference The platform difference between EC2 Cloud and local data center also creates some errors in migrating applications. These errors can be classified into three levels: security, communication, and incorrect instance operation. In our experiment, when the applications are hosted in local data center, the machines are protected by the firewalls, and thus even the operators set simple passwords, the security is complemented by the firewalls. However, when the applications are migrated into the public Cloud, the machine can experience all kinds of attacks and thus too simple passwords may render the virtual hosts in security threats. The second level of this type error is that after the applications are migrated into EC2 Cloud, the operators still set the communication as applications are hosted in the local data center. For example, for operator in one virtual instance to ssh another virtual instance, the identify file which is granted by Amazon must be provided. Without the identify file, the communication within virtual instance can not be set. The third level is rooted in the difference between infrastructures. In the experiments, there were operators who terminated an instance but with actual intention to stop the instance. In EC2 platform, termination of an instance will lead to the elimination of the virtual instance from Cloud and thus all the applications installed and all the data stored within the virtual instance are lost if data is not backed up in persistent storage

like Amazon Elastic Block storage. Thus, this poses critical risks on the instance operations, because a wrong instance operation may wipe out all the applications and data.

Reliability Error: in order to achieve fault tolerance and performance improvements, many enterprise applications like Hadoop and multi-tiered Internet services, usually replicate its data or components. For example, in Hadoop, data is replicated in certain number of DataNodes, while in multi-tiered Internet services, there may exist multiple application servers or database servers. Thus after the migation, if the replication degree is not set correctly, either the migrated application fails to work correctly or fault tolerance level is compromised. For example, in the experiments, there were cases that operator made errors that set the replication degree more than the number of DataNodes in the system. The reliability error sometimes are latent errors.

Shutdown and Restart: this type of error means that the shutdown or restart operation in the migration process may cause errors if not operating correctly. For example, a common data consistency error is caused if Hadoop is incorrectly shuts down the HDFS. Even seriously, sometimes, shutdown or restart error may compromise the source system. In our experiment, when the dependency graph was not completely updated and the source cluster was not shut down, the destination Hadoop cluster initiated to connect to the source cluster and worked as the client to connect to the source cluster. All the operations issued by the destination cluster actually manipulated the data in the source cluster and thus the source cluster data was contaminated. Such kind of error may create disaster impacts on the source cluster and is dangerous if the configuration errors are not deteced.

Software and Hardware Compatability: this error in Hadoop is less common than RUBiS partly because Hadoop is built on top of Java and thus larger interoperability is achieved and also Hadoop involves less different components than RUBiS.

However, the initial Hadoop version selected by one operator was Hadoop 0.19 which showed bugs in the physical machine. After the operator turned to the latest 0.20.2 version, the issue disappeared.

Access Control and Security: single node Hadoop cluster can be set and migrated without root access. However, because multi-node Hadoop cluster needs to change the network inter-connectivity and solve the local DNS resolution issue, the root access priviledge is necessary. One operator assumed that the root privilege was not necessary for multi-node Hadoop installation and was blocked for the network connectivity issues for about 1 hour and then sought to help for the root privilege.

4.3.3 RUBiS Migration Study

In this experiment, we migrate a RUBiS system with one web server and two application servers and two database servers from the local data center to EC2 Cloud. We introduce the configuration errors present in the experiments below:

Dependency Preservation: similar to Hadoop migration this type of error is the most common error in RUBiS migration. Because RUBiS has more intensive dependency among different components than Hadoop, operators made more configuration errors in the migration. For the different tiers of RUBiS system to run cooperatively, dependency constraints need to be specified explicitly in relevant configuration locales. For example, for each Tomcat server, its relevant information needs to be recorded in the configuration file named " workers.properties" in Apache HTTPD server. And the MYSQL database server needs to be recorded in RUBiS configuration file named "mysql.properties". Thus any error in any of these dependency configuration files will lead to the operation error. In our experiments, operators made different kinds of dependency errors. For example, operator migrated the application but forgot to update the Tomcat server name in workers.properties, and thus although the Apache HTTPD server was running correctly, RUBiS was not

operating correctly because the Tomcat server could not be connected. One operator could not find the configuration file location to update the MYSQL database server information in RUBiS residing in the same host as Tomcat and this led to errors and the operator therefore gave up the installation.

Network Connectivity: because there is less node interoperability in the multi-tiered system, different tiers present less needs on network connectivity, thus the network connectivity configuration errors are less frequently seen in RUBiS migration. One typical error is that when the operator was connecting the Cloud virtual instance, he forgot the identity file.

Platform Difference : this error turns out to be a serious fundamental issue in RUBiS migration. Because sometimes the instance rebooting operation may change the domain name, public IP and internal IP, thus even the multi-tiered service is migrated successfully, a rebooting operation may render the application to service interruption. One operator finished the migration and a few configuration errors were fixed, the application was working correctly in EC2. After we turned off the system on EC2 for one day and then rebooted the service, we found that because the domain name had totally changed, thus all the IP addresses or host names information in configuration files needed to be updated.

Reliability Error: because of the widely used replication in enterprise systems, it is typical that the system may have more than one application server or more than one database servers. One operator spelt the name wrong for the second Tomcat server, but because there remained a working Tomcat server due to replication, the service was still going on without interruption. However, a hidden error was introduced to the system if there were no configuration error detection and correction tools for the migration.

Shutdown and Restart: this error shows that incorrect server start or shutdown operation in multi-tiered services may render the whole service unavailable. For

example, the Ubuntu virtual instance selected for MYSQL tier has a different version of MYSQL database installed by default already. One operator forgot to shut down and remove the default installation first before installing the new version of MYSQL and thus caused errors. The operator spent about half an hour to find out the issuses and fixed it. It was twice that the operator forgot to boot the Tomcat server first before the shutdown operation and thus caused errors.

Software and Hardware Compatability: this type of error also happens frequently in RUBiS migration. The physical machine is 64 bits machine, while one operator selected the 32 bits version of mod_jk which was the component used to forward the HTTP request from Apache HTTPD server to Tomcat server and thus incompatibility issues occured. The operator was stuck for about 2 hours, and identified the version error. After the software version was changed into 64 bits, the operator successfully fixed the error. One operator selected an arbitrary MYSQL version which took about 1 hour for the failed installation and then switched to a newer version and finally successfully installed the MYSQL database server.

Access Control and Security: this type of error also produces frequent occurrences. For example, the virtual instance in EC2 Cloud bears the default feature of all ports closed. To enable the SSH operation possible, the security group where the virtual instance resides must open the corresponding port 22. Also one operator configured the Apache HTTPD server successfully but the Web server was unable to connect through port 80 and it took about 30 mins to identify the restrictions from EC2 documentation. Similar errors also happened for port 8080 which was for accessing Tomcat server. One interesting error is that one operator set up well the Apache HTTPD server, but forgot to set the root directory to be accessible and thus the index.html was not accessible and the operator reinstalled the HTTPD server but still did not find out the error. With our reminder, he identified the error and changed the access permission and fixed the error. Operators also made errors in

granting privileges to different users and solved by seeking help to the MYSQL documentation.

4.3.3.1 Installation Errors

In our experiment, we find that operators may make all kinds of errors in installation or redeployment of the applications in Cloud. More importantly, these errors seem to be common across all the application types. Here we categorize these errors into the following types:

Context information error: this is a very common installation error type and a typical example is that operators forget the context information they aim before the installation. For example, the operators remembered the wrong path to install their applications and have to reinstall the applications from scratch. And also if there are no automatic installation scripts, the same procedures need to be repeated again and again. If the scale of the computing system is large, then the repeated installation process turns out to be a heavy burden for system operators. Thus a template based installation approach is highly demanded.

Environment preparation error : In the migration experiments, before any application can be installed, the prerequisite resource needs to be ensured. For example, there were migration failures created due to the small available disk space in virtual instance in migrating RUBiS. A similar errors is that the operator created a virtual instance with 32 bits operating system, while the application was a 64 bits version. Thus, it is necessary to check the environment before the application installation starts. An automatic environment checking process helps to reduce the errors caused by incorrect environment settings.

Prerequisite resource checking error : this error is originated from the fact that every application depends on certain set of prerequisite facilities. For example, the installations of Hadoop and Tomcat server presume the installation of Java. In

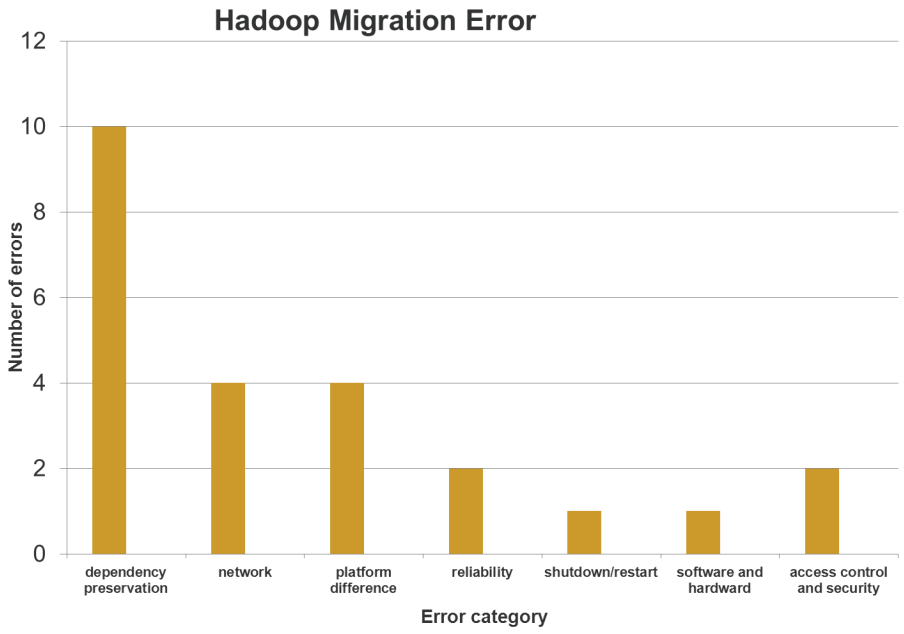

Figure 28: Hadoop migration error

the experiment, we found the migration or installation process were prone to be interrupted by the ignorance of installing prerequisite standard facilities. For example, the compilation process needs to restart again due the lack of "gcc" installation in the system. Thus, a check on the prerequisite resources or facilities helps to reduce the interruptions of the migration process.

Application installation error: this error is the most common error type experienced by the operators. The concrete application installation process usually consists of multiple procedures. We found that the operator made a lot of repeated errors even when the installation process for the same application was almost the same. For example, operators forgot the building location of the applications. Thus a template based application installation process will help to faciliate the installation process.

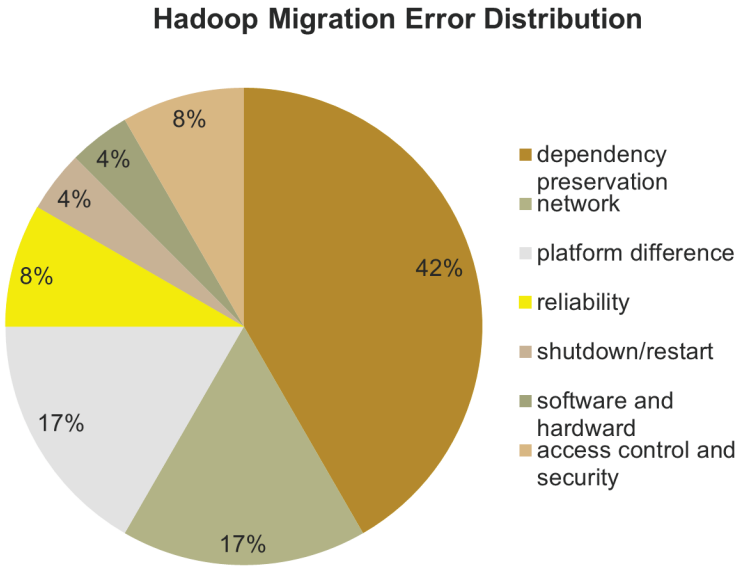

Figure 29: Hadoop migration error distribution. The legend lists the error types in the decreasing frequency order.

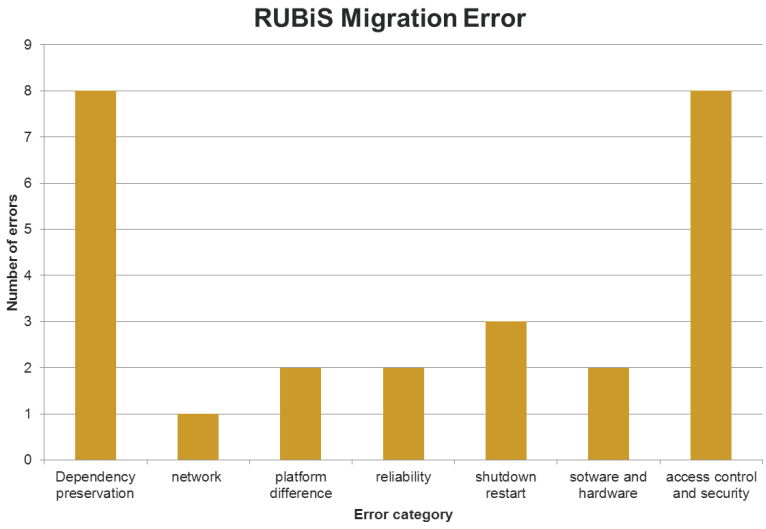

Figure 30: RUBiS migration error

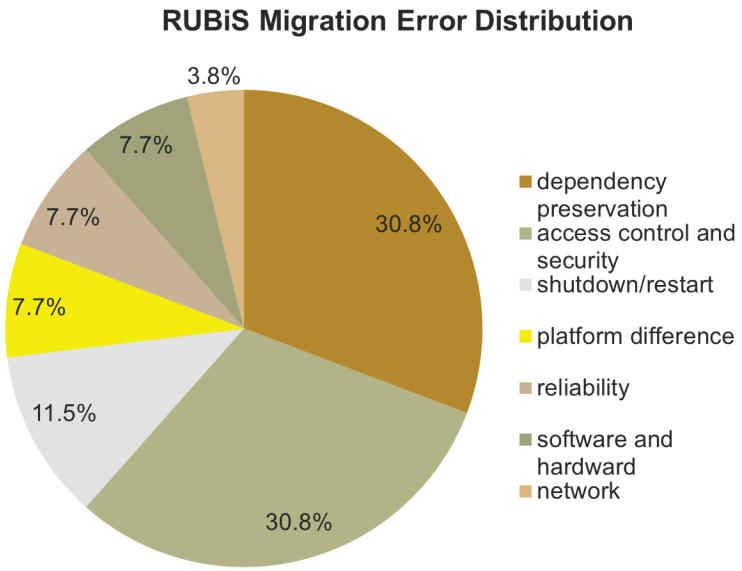

Figure 31: RUBiS error distribution. The legend lists the error types in the decreasing frequency order.

4.3.4 Summary

In this section, we introduce the error distributions for each specific application and the error distribution across the applications.

Figure 28 and Figure 29 show the number of errors and particular error distribution in Hadoop migration experiment. In Figure 28, the X-coordinate indicates the error types as we analyzed in the previous sections and the Y-coordinate shows the number of errors for the particular error type. Figure 29 shows the shares of each error type in the total number of errors. In this set of experiments, there were totally 24 errors and among them the dependency preservation error happened most frequently. 42% of the totall number of errors belong to this error type with 10 occurrences. Operators made all the four levels of dependency preservation errors. These kinds of errors took long time for operators to detect. For example, an incomplete dependency constraint checking error took the operator two and a half hours to identify the cause of the error and fix it. Network connectivity error and platform difference

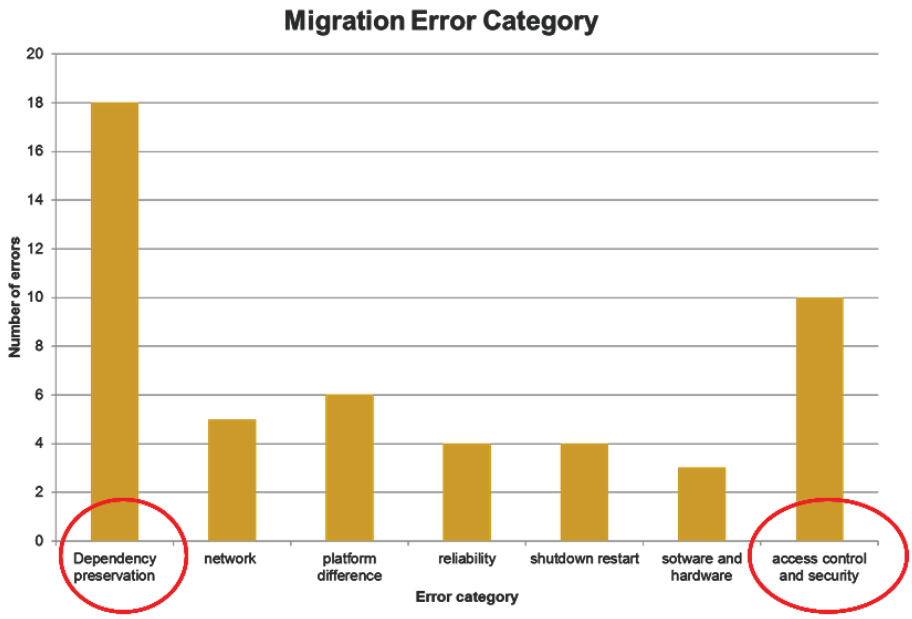

Figure 32: Overall migration error

error were the next most frequent error types, each taking 17% of the total erros. Network connectivitiy errors included local DNS resolution and IP address update errors. One typical platform difference error was that the termination of an instance led to the data loss. Totally these three types of errors takes 76% of the total errors, dominating the error occurrences.

Figure 30 and Figure 31 show the number of error occurrences and error distribution for RUBiS migration experiment respectively. There were totally 26 error occurrences in this process. The dependency preservation error and access control and security were the two most frequent error types, each taking 31% of the total erorrs with 8 occurrences. Together, both error types covered 62% of all the errors and dominated the error occurrences. The distribution of errors in RUBiS migration experiment was very different from the distribution of Hadoop migration experiment. For example, the number of access and security errors in RUBiS was 4 times the number of this type error in Hadoop migration. This is because RUBiS migration demanded the correct access control settings for many more entities than Hadoop. And

the majority of the access control errors were file access permission errors. Because changing the file access permission is a common operation in setting web service, sometimes operators forgot to validate whether the access permissions were set correctly or not. Also when there were errors and system could not run correctly, the operators ignored the possibility of this type of simple error and thus led to long error identification time. For example, one of this type error took more than 1 hour to identify. Also there were more ports to open in RUBiS migration than Hadoop migration and this fact also led to the high frequency of access control errors in RUBiS migration. RUBiS migration presented more software/hardware compatibility errors than Hadoop migration because more different components were involved in RUBiS application and for similar reasons, there were more "shutdown/restart" errors in this migration. On the other hand, Hadoop migration presented to have more network connectivity errors and platform difference errors than RUBiS migration, which is because Hadoop nodes require more intense connectivities than the nodes in RUBiS, for example, the master node needs to have passwordless accesses to all the slave nodes.

Figure 32 and Figure 33 respectively show the number of error occurrences and error distribution summarized across Hadoop migration and RUBiS migration experiments. As one can see, dependency preservation error created the most frequent occurrences and it accounted for 36% of the total errors and in practice it was also the type of error that on average took the longest time to identify and fix. Because dependency constraints are widely distributed among system configurations, it is very prone to be broken by common configuration changes. The second biggest error source was "access control and security" error which accounted for 20% of the total number of errors. It was very easy for operators to change the file permissions to incorrect settings or some other habits which were fitting in local data center might render the application under security threats in the Cloud environment. The difference between

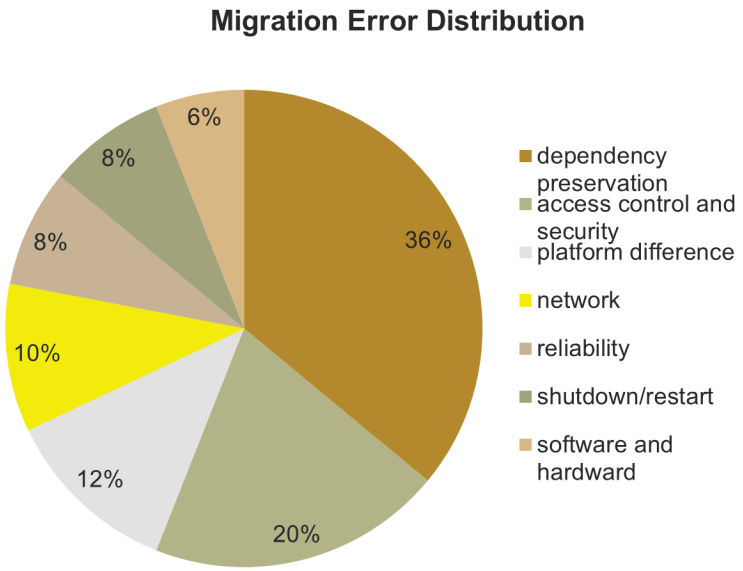

Figure 33: Overall migration error distribution. The legend lists the error types in the decreasing frequency order.

Cloud and local data centers formed the third largest source of error, accounting for 12% of all the errors. Many common operations in local data center might lead to errors if no adjustments to Cloud environment were made. These three types of errors dominated the error distribution which accounted for 68% of the total errors. In addition to these errors, network connectivity was also one important source of errors, accounting for 10% of the total errors, because of the heavy inter-nodes operations in today's applications. The rest errors accounted for 22% of the total errors. These error distributions build a solid testbed for us to test the later proposed migration approach. This motivates us that the first aid needed from configuration management is to reduce the dependency preservation errors. It is highly demanded that a configuration management system should not only provide a mechanism to reduce the configuration errors, but also equip the system with active configuration error detection.

4.4 CloudMig Architecture Overview

The experimental study showed that the installation mistakes and configuration errors were the two major sources of errors in migrating applications from local data centers to Cloud. Thus a migration management framework which provides the installation automation and configuration validation is highly demanded. We propose the"CloudMig" system which utilizes "template" to simplify the large scale enterprise system installation process and utilizes "policy" as an effective approach to validate the configuration and monitor the configuration changes. In this section, we overview the architecture of CloudMig system, which aims to coordinate the migration process across different data centers, simplify the migration process through template, uses policy to promote migration assurance and monitors the configuration changes or updates.

CloudMig configuration management and validation system mainly consists of four parts: a central configuration operator server, local configuration management engine, configuration template tool and policy tool. In the next two sections, we mainly discuss the template model which lays the foundation for automatic installation component and configuration policy model which is the basis for policy based configuration management component.

4.5 Template Model

CloudMig uses template as an effective solution to simplify the installation process. Template is a pre-formatted example file containing place holders for dynamic information to be substituted for concrete use.

In CloudMig, the installation and configuration management is operating in the unit of application. That is, each application corresponds to a template set and a validation policy set. The central management server is responsible to manage the collection of templates and configurations per application basis and provides migration

planning for the migration process.

As one can see in the experiments, one big obstacle and source of errors in application migration is the deployment process which is also a recurring process in system deployment or migration. We propose to use the template approach to reduce the complexities of the installation process and reduce the chances of errors. Installation template is installation script with place holders for dynamic information. Templates simplify the recurring installation practice of particular applications by substituting the dynamic information with new values. For example, in an enterprise system with 100 nodes, there will be plenty of applications: MYSQL database, Tomcat server, Hadoop, etc. Distributed application may span many nodes, for example, a Hadoop application may involve hundreds of nodes to achieve scalability. For each application, its installation templates are stored in the installation template repository which is sorted per application unit. The intuitive idea of template is that through information abstraction, the template can be used for many similar nodes with only the parameter substitution efforts needed to simplify the installation process for large scale systems. For example, if the Hadoop system consists of 100 DataNodes, then only a single installation template is stored in the installation template repository and each DataNode will receive the same copy of installation template with only parameter substitution efforts needed before running the installation scripts to set up the DataNode component in individual node. CloudMig classifies the templates into the following types:

1. Context dictionary: is the template specifying the context information about the application installation. For example, the installation path, the preassumed Java package version, etc. Context dictionary template can be as simple as a collection of the key-value pairs. Users specify the concrete values before a particular application installation. Dynamic place holders for certain key context information achieve the installation flexibility and increase the ability

to find out the relevant installation information in system failures.

2. Standard facility checklist template: is the script template to check the prerequisites to install application. Usually these are some standard facilities such as Java or OpenSSH and etc. Typical checklists includes: verifying the Java path setting, installation package existence and etc. These checklists are common to many applications and are prerequisites for the success of installing the applications and thus running a check before the actual installation can effectively reduce the errors caused by ignorance of the checklist items. For example, both Hadoop and Tomcat server rely on the correct Java path setting and thus the correct setting of Java path is the prerequisite of successfully installing these two applications. In CloudMig, we propose to accumulate such templates into a library which is shared by multiple applications. Running the checklist check can effectively speed up the installation process and reduce the errors caused by carelessness on prerequisites.

3. Local resource checklist template: is the script template to check the hidden conditions for an application to be installed. A typical example is like enough available disk space quota. Similarly such resource checklist templates are also grouped into the library to reduce the installation errors or installation halt.

4. Application installation template: is the script template used to install a particular application. The context dictionary is included or sourced into this type of application as context information. Organizing installation templates simplifies the installation process and thus reduces the overheads in recurring installations and migration deployments.

4.6 Configuration Policy Model

In this section, we introduce the basic concept of "Configuration Policy" which plays the key role in monitoring and detecting configuration anomalies in large enterprise application migration. Further, we discuss "Continual Query" based Configuration Policy as an effective configuration policy model used in CloudMig framework.

4.6.1 Configuration Policy Concept

Configuration files for applications usually specify the settings of system parameters, layout the system components dependencies and thus impact directly the way system is running. As enterprise applications scale, the number of components increase rapidly and the correlations among system components evovles with added complexity. In term of complexity, configuration files for a large system may cover many aspects ranging from host system information, to network setting such as port number, to security protocol and etc. Any typo or error may disable the operational behavior of the whole application, as we analyzed in the previous experiments. Configuration setting and management is usually a long term practice, starting from the time the application is set up until the application is ceased to use. During this long time practice, different operators may be involved in the configuration management practices and operate on the configuration settings based on their understandings, thus it further increases the probability of errors in configuration management. In order to fully utilize resources, enterprises may bundle multiple applications in the same machine, the addition of new applications may necessitate the need to change the configurations of existing applications. Security threats such as virus, also pose the demands of effective configuration monitoring and management.

In CloudMig, we propose policy as an effective approach in ensuring the constraints of configurations to be complied. A "policy" is a specialized tool to specify

the constraints on the configuration of a certain application. It specifies the constraints that the application configuration must conform in order to assure the whole application is configured correctly. For example, in Hadoop system, a sample policy is like " the replication degree can not exceeds the number of DataNodes". We will introduce the policy concept in more details in later sections. The basic idea is that if operators are equipped with a tool to define the constraints that configuration must follow and run the policy checks with a certain frequency, then although the errors are unavoidable, running the policy checks helps to detect the errors and eliminate the errors. Here are a few policy examples that operator may have in migrating Hadoop system.

1. The replication degree can not be larger than the number of DataNodes
2. There is only one master node
3. The master node of Hadoop cluster should be named "dummy1"
4. The task tracker node should be named "dummy2"

As the system evolves and the configuration repository increases, performing such checking mannually will become a heavy and error-prone process. For example, in the enterprise Internet service systems, there may be hundreds of nodes, and the configuration of each node needs to follow certain constraints. For example, for load balancing purpose, different Apache HTTPD servers corresponds to different set of Tomcat servers. Incorrect setting of relevent configuration entries will directly lead to an unbalanced system and even cause the system to crash when workload burst happens. Thousands of configuration entries, hundreds of nodes, and many applications makes the mannual configuration correctness checking and error correction impractical. Thus an automatic configuration constraint checking framework can greatly simplify the system management of large scale enterprise systems and thus is

highly demanded. In CloudMig, we aim to provide a configuration validation framework which can automate the configuration validation process and thus reduce the complexity of migrating application from local data center to Cloud.

4.6.2 Continual Query Based Policy Model

In CloudMig, we propose continual query based policy model as the concrete realization form of policy checking. A continual query (CQ) [80] is a triple in the form of CQ(Query, Trigger, Stop) in which Query is query format expression on data, and triggers the condition to run the CQ and Stop is the condition to terminate the execution of CQ. Once a CQ is installed, it will run the query as long as the trigger condition is met. Trigger condition can be different types such as time-based trigger and content-based trigger. Examples of time-based trigger are like: "run the CQ every 5 mins" or "at 8pm, run the CQ". Examples of content-based trigger are like "free disk space is less than 1G, run CQ."

In the practice of configuration validation, the configuration information can be organized in different ways, such as key-value maps, XML format, or even a database. Thus we flexibly design the concrete format of Contiual Query Policy (CQP) in the setting of CloudMig to be the following tuple: CQP(policyID, appName, query, trigger, action, stopCondition). Each component is defined as:

1. *policyID* is the unique numeric identifier of the policy.

2. *appName* is the name of the application that is installed or migrated to the host machine.

3. *query* is the content of the policy checking. It can be a Boolean expression upon a simple key-value pair repository or SQL-like query or XQuery on a relational database.

4. *trigger* is the condition that the query will be executed. It can be classified into

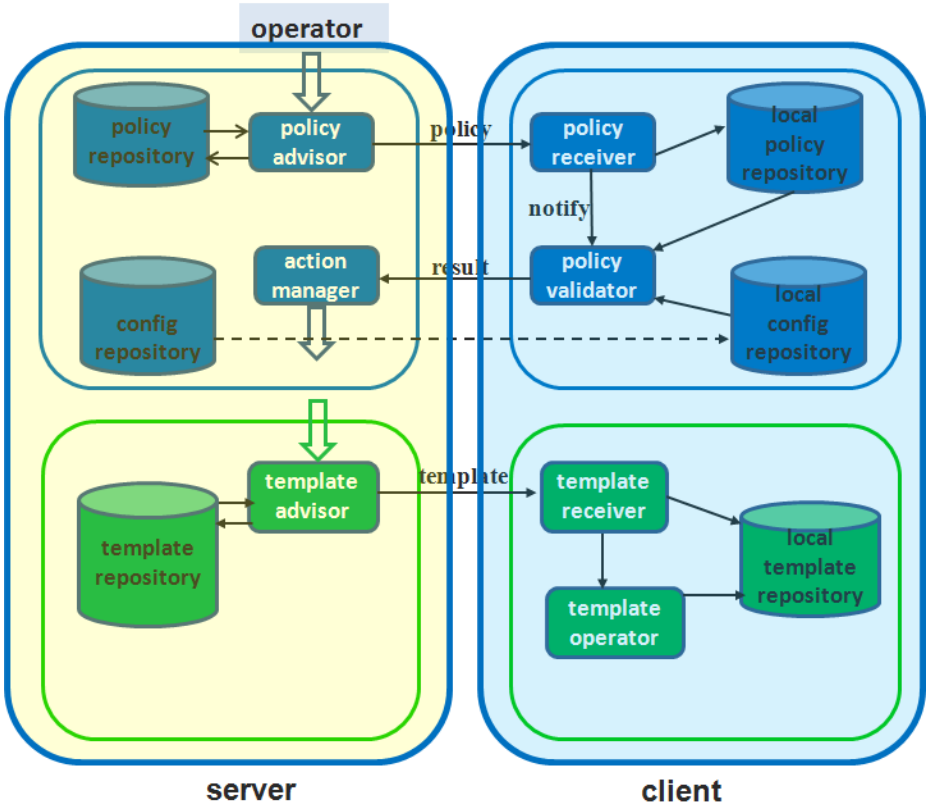

Figure 34: CloudMig Architecture Design

time-based condition or content-based condition. In the existing prototype of CloudMig, we mainly use time frequency based trigger.

5. *action* indicates the action that will be taken upon the query results. It can be a warning flash in configuration tables or an email warning message.

6. *stopCondition* is the condition upon which the CQP will stop to execute.

4.7 Architecture Details

4.7.1 CloudMig Template and Policy Management Server

CloudMig is aiming to manage the installation template and configuration policy and simplify migration for large scale enterprise systems which may scale to thousands of nodes with large number of applications. Each application owns its configuration set such that the whole system amounts to a large sets of configurations. A central

management system helps to manage the large scale templates and configuration policies more effectively and provides administrators with convenience to operate on the templates and policies. Detaching the template and policy management from individual application and utilizing a centralized server also achieves the reliability in the case individual node fails.

In CloudMig, the management server operates at the unit of a single application. That is, each application corresponds to an installation template set and a configuration validation policy set. The central management server is responsible to manage the large scale configurations per application basis and provides migration planning for the migration process. More concretely, the configuration management server mainly consists of two components:

4.7.1.1 Installation Template Management Engine

As shown in Figure 34, installation template management engine is the system component which is responsible to create template, update template, advise the template for installation. It consists of a central template repository and a template advisor. Template Repository is the store for the template collections of all the applications for the computing system. Template advisor provides the operators with the functionalities including creating, updating, deleting templates upon the template repository. On per application basis, operators create application template set, add new templates to the set, update templates from this set or delete templates. Template advisor assumes the job to dispatch templates for new installations and transmit updates to templates. For example, during the process of RUBiS installation, for a specific node, template advisor dispatches the appropriate template depending on the server type (web server, application server or database server) and transmits the new installation set to the particular node.

More concretely, for each application, the central installation template management engine builds the context library which stores all the source information in the key-value pairs, and selects a collection of standard facility checklist templates which apply to the particular application, and pick a set of local resource checklist templates as the checklist collection for the application, and finally builds the specific application installation template. The central management engine then bundles these collections for the particular application and transmits to the client installation template manager to start installation instance.

4.7.1.2 Configuration Policy Management Engine

Policy engine is the central management unit for the policies. Policy engine consists of four components: policy repository, configuration repository, policy advisor, and action manager. Together they cooperate to provide the service to create, maintain,dispatch, monitor policies and execute the corresponding actions on the policy execution results. More concretely, the different components of policy engine are introduced below:

1. The policy repository is the central store where all the policies for all the applications in the computing system are maintained. It is also organized based on the per application basis. Each application corresponds to a specific set of policy set. This policy set is open to addition, update, or deletion operations. Each policy corresponds to a constraint set on the application.

2. The policy advisor works on the policies in the policy repository directly and provides the functionalities for application operators to express the constraints into policy. Application operators creates policies through this interface.

3. The "config repository" stores all the configuration files based on per application basis. It transmits the configurations to the local config repository.

4. The action manager handles the validation results from policy validator running in client and triggers the corresponding action on certain result, in the form of alert message, alert email and etc.

4.7.2 CloudMig Management Client

The CloudMig configuration management client is running in each node in the computing system, which is responsible to manage the configurations in the node locally. Corresponding to the CloudMig server, CloudMig client works as a thin local manager for the templates and policies which only apply to a particular node. A client mainly consists of two components: client template manager and client policy manager. Next we will introduce these two components in details.

4.7.2.1 Client Template Manager

Client template manager manages the templates for all the applications installed in the host node on per application basis. It consists of three components: template receiver, template operator and local template repository. The template receiver receives the templates from remote CloudMig Configuration Management Server and delivers the templates to local template operator. The local template operator sets up the application based on the template with necessary substitution operations. The local template operator is also responsible to manage the local template repository which stores all the templates for the applications that reside in this node.

The concrete process of template based installation works as follows: after the client template manager receives the collection of installation templates from Installation Template Management Engine in Central server, it will run the local resource checklist templates first to detect if there are any prerequisite checklist items which are not met. For example, it checks if the available disk space is less than the amount needed to install the application, or if the user has the access permissions to the installation path etc. Next, the standard facility checklist template will run to detect

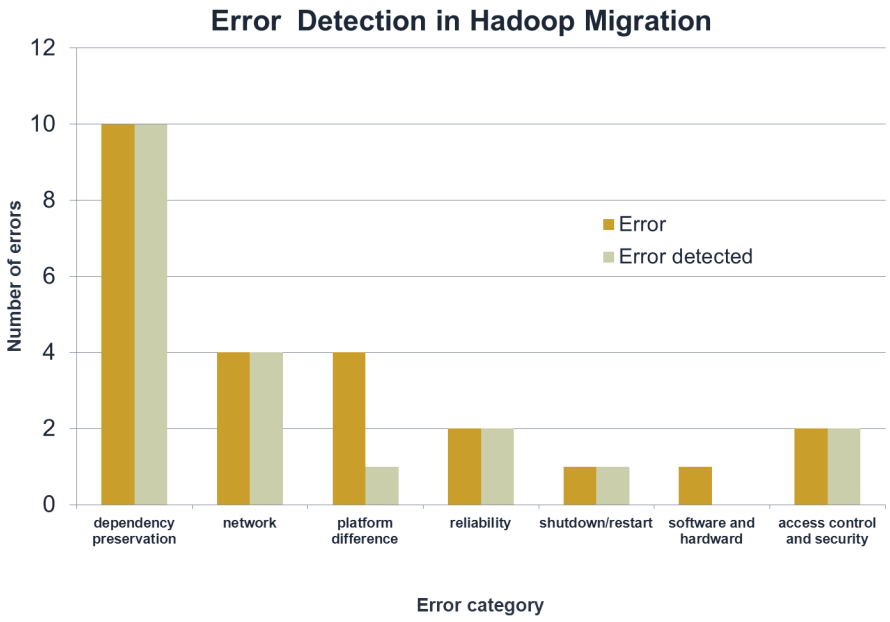

Figure 35: Hadoop error detection

if all the standard facilities are installed or not. Finally, the dynamic information in application specific templates are substituted and the context dictionary is integrated to run this normal installation process.

4.7.2.2 Client Policy Manager

There is a client policy manager residing together with the host node to manage the policies for the local node. It mainly consists of policy receiver, policy validator, local policy repository and local config repository. The policy receiver receives the policies transmitted from policy advisor in central policy server, and stores the policies in the local policy repository. The local config repository receives the configuration data directly from the central config repository. The local policy validator runs each policy. It retrieves the policy from local policy repository and searches the related configuration data to run the policy upon the configuration data. The policy validator transmits the validation results to the action manager in the central server to take the alert actions.

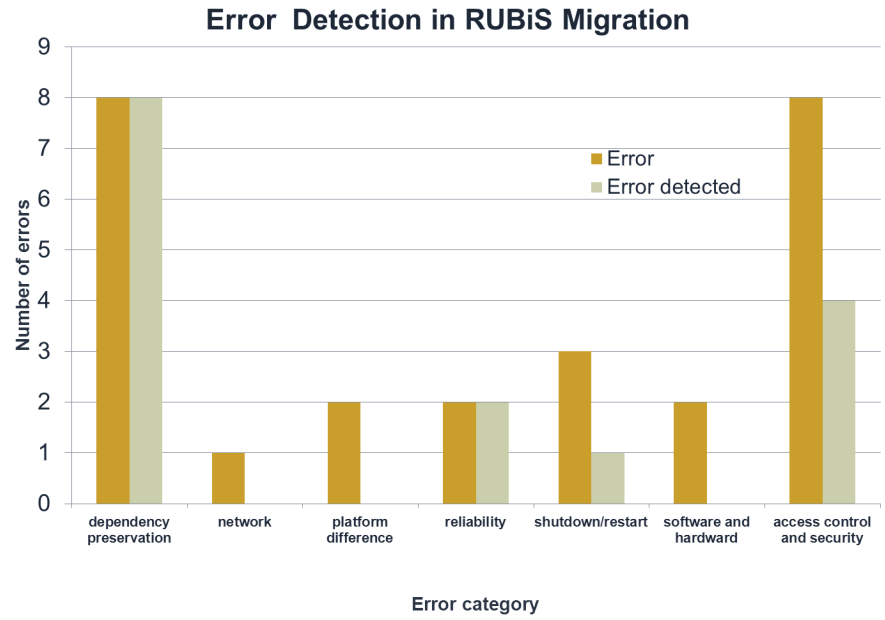

Figure 36: RUBiS error detection

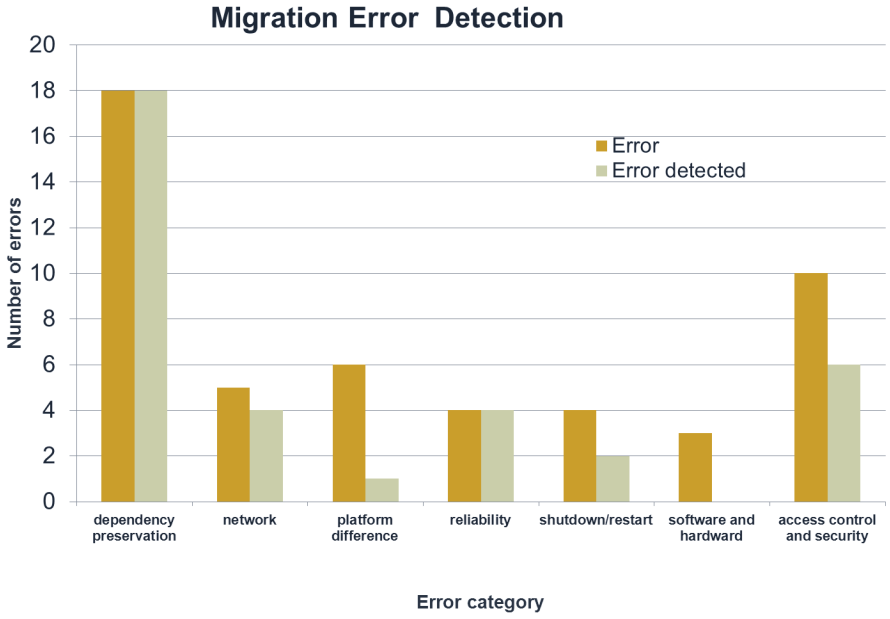

Figure 37: Overall migration error detection

Table 2: Migration Error Detection Rate

Migration Type	Error Detection Rate
Hadoop migration	83%
RUBiS migration	55%
all migrations	70%

4.8 Experiments

We run CloudMig with the same operator on the same experiment, after the manual migration process is done. We count the number of errors that can be deteced by CloudMig configuration management and installation automation system. CloudMig overall achieves high error detection rate. Below we will discuss some of the experimental results.

Figure 35 shows the error detection results for Hadoop migration experiment. As one can see that the configuration checking system can detect all the dependency preservation errors, network connectivity errors, shutdown restart errors, and all the access control errors. This confirms the effectiveness of the proposed system, in that it can detect the majority of the configuration errors. The two types of error that can not be fully detected are platform difference error and software/hardware compatibility errors. For platform difference errors, this is because the special property of the platform difference error requires the operators to understand the uniqueness of the particular Cloud platform fully first. As long as the operator understands the platform sufficiently, such errors can be reduced significantly as well. The reason that current implementation of CloudMig can not detect software/hardware compatibility errors notably is because the default configuration data in the application does not contain intense software/hardware compatibility information, such that external human operator involvements are necessary to designate the policies. In this phase of implementation, we mainly focus on the configuration checking triggered by the original configuration data. We reasonably believe that as operators weave more compatibility policies, such type of errors can be reduced significantly. As Table 2 shows,

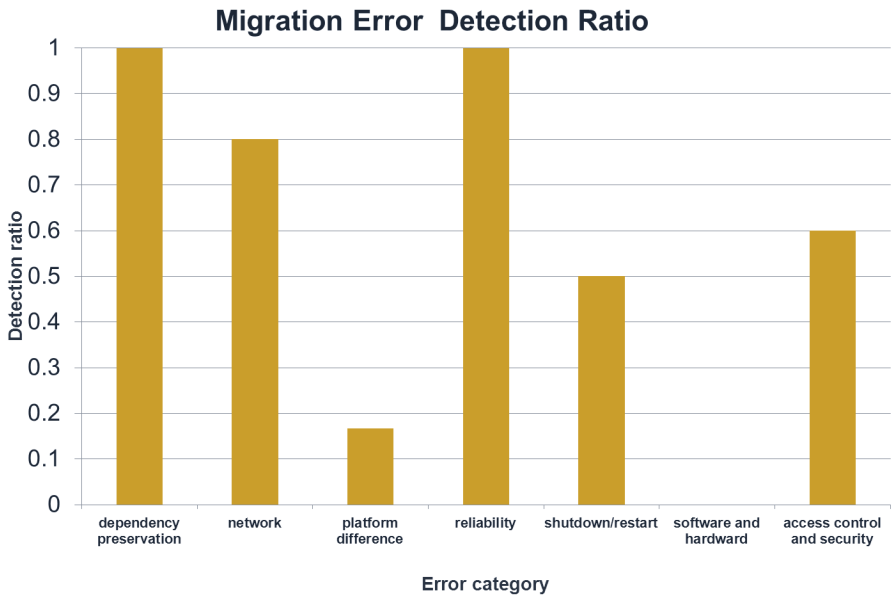

Figure 38: Overall migration error detection ratio

totally CloudMig could detect 83% of the errors in Hadoop migration.

Figure 36 shows the error detection result for RUBiS migration experiment. In this experiment, we can see that CloudMig can detect all the dependency preservation errors and reliability errors. However, because multi-tiered Internet service system involves more different components, and more complicated software/hardware compatibility issues are created compared with the case of Hadoop system and we are focusing on the configuration driven by the default configuration data, CloudMig system did not detect the software/hardware errors. This also shows that in the migration process, the operators are suggested to put special attention to the software/hardware compatability issues, otherwise such errors are difficult to be detected by automated tools. The reason that the proposed tool detectes half of the access control errors in RUBiS is because these errors include MYSQL privileges grant operations which are embedded in the application itself and the proposed tools can not intervene the internal operations of MYSQL. As shown in Table 2, CloudMig can detect 55% of the errors in RUBiS migration.

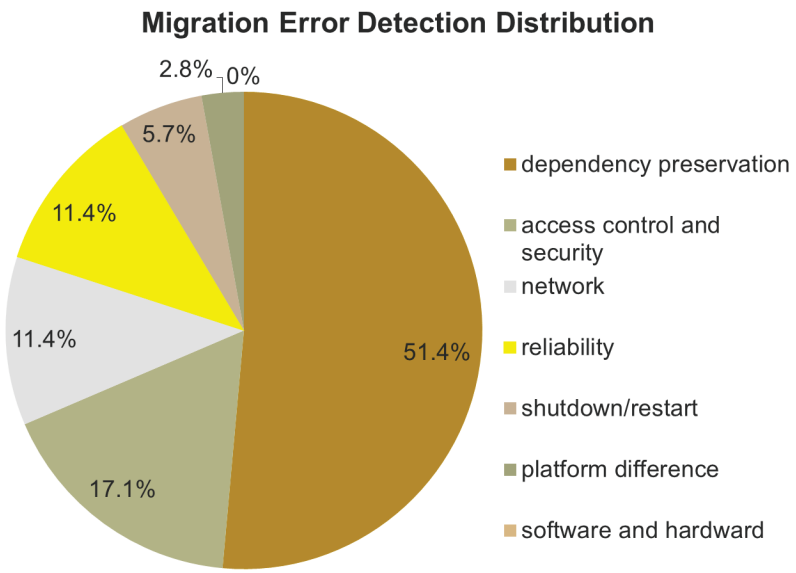

Figure 39: Overall migration error detection percentage. The legend lists the error types in the decreasing percentage order.

Figure 37 and Figure 38 respectively show the number of detected errors and error detection ratio towards each error type summarized across the Hadoop migration experiment and RUBiS migration experiment. Overall, CloudMig can detect all the dependency preservation and reliability errors and 80% of the network errors and 60% of the access control and security errors. In total, these four types of errors accounted for 74% of the total error occurrences. For shutdown/restart errors, CloudMig detected 50% of such errors and did not detect the software/hardware compatibility errors. This is because the application configuration data usually contains less information related with shutdown/restart operations or software/hardware compatibility issues and this fact makes the configuration checking on these types of errors difficult without external extra policy involvemenT. Figure 39 shows the the percentage of errors in the total number of detected errors. One can see that 51% of the detected errors are dependency preservation errors, and 17% of the detected errors are network errors. Table 2 shows that totally across all the migrations, the error detection rate is 70%.

Overall these experimental results show the efficacy of CloudMig in reducing the migration configuration errors and simplifying the migration process.

4.9 Conclusion

In this chapter, we discuss the system migration challenge faced by enterprises in the process of migrating local data center applications to the Cloud platform. We analyze why such migration is a complicated and error-prone process and pointed out the limitations of the existing approaches to solve this problem automatically. Then we introduce the operator experiment study we have conducted to investigate the error sources. From these experiments, we build the error classification model and analyze the demands for a configuration management system. Based on the operator study, we design the CloudMig system, which is a continual query based configuration policy checking system, in that operators weave important configuration constraints into continual query policies and periodically run these policies in the system to monitor the configuration changes and the possible configuration constraints violation. CloudMig consists of two components: the continual query based policy checking system and the template based installation automation system aiming to help operators to simplify the installation errors. The experiments show that CloudMig can effectively detect the majority of the configuration errors in the migration process.

CHAPTER V

CONCLUSION AND FUTURE WORK

In this dissertation we have studied three important research problems in data and application migration. In this concluding chapter, I will first summarize the contributions of this dissertation and then give a brief overview of the open issues and challenges ahead.

5.1 Main Contributions

This dissertation has made technical contributions in the context of data migration, service migration and application migration.

`Data Migration:` The main contribution we have made in the data migration area is to investigate the opportunity of automated data migration in multi-tier storage systems. The significant IO improvement in Solid State Disks (SSD) over traditional rotational hard disks (HDD) motivates the integration of SSD into existing storage hierarchy for enhanced performance. We developed adaptive look-ahead data migration approach to effectively integrate SSD into the multi-tiered storage architecture. When using the fast and expensive SSD tier to store the high temperature data (hot data) while placing the relatively low temperature data (low data) in the HDD tier, one of the important functionality is to manage the migration of data as their access patterns are changed from hot to cold and vice versa. For example, workloads during day time in typical banking applications can be dramatically different from those during night time. We designed and implemented an adaptive lookahead data migration model. A unique feature of our automated migration approach is its ability to dynamically adapt the data migration schedule to achieve the optimal migration effectiveness by taking into account of application specific characteristics and I/O

profiles as well as workload deadlines. Our experiments running over the real system trace show that the basic look-ahead data migration model is effective in improving system resource utilization and the adaptive look-ahead migration model is more efficient for continuously improving and tuning of the performance and scalability of multi-tier storage systems.

`Service Migration:`: The main contribution we have made in the area of service migration in this dissertation research is to address the challenge of ensuring reliability and balancing loads across a network of computing nodes, managed in a decentralized service computing system. Considering providing location based services for geographically distributed mobile users, the continuous and massive service request workloads pose significant technical challenges for the system to guarantee scalable and reliable service provision. We design and develop a decentralized service computing architecture, called Reliable GeoGrid, with two unique features. First, we develop a distributed workload migration scheme with controlled replication, which utilizes a shortcut-based optimization to increase the resilience of the system against various node failures and network partition failures. Second, we devise a dynamic load balancing technique to scale the system in anticipation of unexpected workload changes. Our experimental results show that the Reliable GeoGrid architecture is highly scalable under changing service workloads with moving hotspots and highly reliable in the presence of massive node failures.

`Application Migration:` The third research contribution in this dissertation research is focused on studying the process of migrating applications from local physical data centers to Cloud. We design migration experiments and study the error types and further build the error model. Based on the analysis and observations in migration experiments, we propose the CloudMig system which provides both configuration validation and installation automation and effectively reduces the configuration errors and installation complexity.

5.2 Open Issues and Future Research Directions

This dissertationdissertation has presented a set of architectural designs and technical developments towards improving performance and utilization for data migration, improving fault tolerance and load balance for service migration, and improving efficiency and effectiveness of application migration by reducing migration configuration errors at operator levels for complex systems such as RUBiS and Hadoop. The inherent problems in efficient migration of data, services and applications remain to be challenging and deserving continued research and investigation. In this section I will summarize three open issues that are directly related to the research development of this dissertation.

`System Management in Cloud:` in this dissertation, we have proposed the architectural design and technical development towards simplifying the overheads in migrating applications from local data center to Cloud. As more and more enterprises migrate their systems to Cloud platform, how to manage the systems in Cloud effectively poses an important challenge. Compared with traditional systems, systems deployed within or across the Clouds have to meet many more challenges, ranging from deployment complexity, to security, to quality of performance, to system integration, to availability and reliability. As a result studying more generic system management in the context of Cloud is an important extension to the application migration problem we solved in this dissertation.

`Service Level Agreement Requirement (SLA) Guaranteed Storage Systems:` In this dissertation, we addressed the problem of data migration in the context that different workloads are operated by the same enterprise entity. As storage systems are shared by more and more applications or users in the Cloud computing environment, different applications from different users pose different Service Level Agreement requirements (SLA) on the storage component. A key to meeting the SLA requirements is to exploit the IO efficiency of different storage devices such as SSD and HDD and

take advantage of the access patterns of different applications. A direct extension to the data migration work in this dissertation is to start from building the SLA model for multitiered storage system supporting many Cloud users and then study the adaptive data migration and replication scheme which optimizes the SLA goal.

`Improving Availability and Reliability:` To improve system performance and reliability,distributed computing systems like Hadoop utilize replication and migration. In the multi-tenant environment of Cloud, multiple users or co-existing applications present different reliability and availability requirements. One important challenge is to develop fault tolerance algorithms to effectively and reliably manage resources among different applications to meet differentiated SLA goals and business objectives.

REFERENCES

[1] "Amazon EC2." http://aws.amazon.com/ec2/.

[2] "Cloud Migration." http://www.opencrowd.com/services/migration.php.

[3] "EMC Delivers Next Generation Of Enterprise Flash Drives." `http://www.netapp.com/us/products/storage-systems/fas3100/`.

[4] "Hadoop project." http://hadoop.apache.org/.

[5] "IBM DS8000." `http://www-03.ibm.com/systems/storage/disk/ds8000/`.

[6] "MYSQL Database." http://www.mysql.com.

[7] "NetApp FAS3100 System." `http://www.netapp.com/us/products/storage-systems/fas3100/`.

[8] "Office Cloud." http://www.officetocloud.com.

[9] "RUBiS benchmark." http://rubis.ow2.org/.

[10] "The Flash Cache Alternative to SSDs." `http://www.hds.com/pdf/Hitachi-Universal-Storage-Platform-V-ESG-Brief-0507.pdf`.

[11] "Computer and Internet use in the United States: 2003." http://www.census.gov/prod/2005pubs/p23-208.pdf, July 2006.

[12] "JMS." http://java.sun.com/products/jms/, July 2006.

[13] "Live Meeting." http://www.microsoft.com/office/livemeeting, July 2006.

[14] "Skype." http://www.skype.com, July 2006.

[15] "Bittorrent." http://www.bittorrent.com, July 2009.

[16] Aberer, K., Cudr-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., and Schmidt, R., "P-grid: A self-organizing structured p2p system," *SIGMOD Record*, vol. 32, pp. 29 – 33, September 2003.

[17] Agarwal, P. K., Arge, L., and Erickson, J., "Indexing moving points," in *Proceedings of ACM PODS*, 2000.

[18] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R., "Design tradeoffs for ssd performance," in *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, (Berkeley, CA, USA), pp. 57–70, USENIX Association, 2008.

[19] Aguilera, M. K., Keeton, K., Merchant, A., Muniswamy-Reddy, K.-K., and Uysal, M., "Improving recoverability in multi-tier storage systems," in *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Washington, DC, USA), pp. 677–686, IEEE Computer Society, 2007.

[20] Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E., "A survey on sensor networks," *IEEE Communication Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[21] Alsberg, P. and Day, J., "A principle for resilient sharing of distributed resources," in *Proceedings of ICSE*, 1976.

[22] Alsberg, P. A. and Day, J. D., "A principle for resilient sharing of distributed resources," in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, (Los Alamitos, CA, USA), pp. 562–570, IEEE Computer Society Press, 1976.

[23] Amir, Y. and Tutu, C., "From total order to database replication."

[24] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R., "Resillient overlay networks," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[25] Anderson, E., Hall, J., Hartline, J., Hobbs, M., Karlin, A. R., Saia, J., Swaminathan, R., and Wilkes, J., "An experimental study of data migration algorithms," 2001.

[26] Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., and Veitch, A., "Hippodrome: Running circles around storage administration," in *In Proceedings of the Conference on File and Storage Technologies*, pp. 175–188, USENIX Association, 2002.

[27] Aron, M., Sanders, D., Druschel, P., and Zwaenepoel, W., "Scalable content-aware request distribution in cluster-based network servers," in *USENIX Annual Technical Conference*, 2000.

[28] Babaoglu, O., Davoli, R., Montresor, A., and Segala, R., "System support for partition-aware network applications," in *International Conference on Distributed Computing Systems*, pp. 184–191, 1998.

[29] Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B., and Khuller, S., "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proceedings of INFOCOM*, 2003.

[30] Banerjee, S., Bhattacharjee, B., and Kommareddy, C., "Scalable application layer multicast," in *Proceedings of the 2002 ACM SIGCOMM Conference*, 2002.

[31] Baset, S. A. and Schulzrinne, H., "An analysis of the skype peer-to-peer internet telephony protocol," Tech. Rep. cucs-039-04, Department of Computer Science, Columbia University, September 2004.

[32] Brewer, E. A., "Towards robust distributed systems (abstract)," in *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, (New York, NY, USA), p. 7, ACM, 2000.

[33] Bu, T. and Towsley, D., "On distinguishing between internet power law topology generators," in *IEEE INFOCOM*, (New York, NY), IEEE, June 2002.

[34] Carzaniga, A. and et al., "A routing scheme for content-based networking," in *INFOCOM*, 2004.

[35] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L., "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.

[36] Castro, M., Druschel, P., Kermarrec, A., and Rowstron, A., "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[37] Chawathe, Y., *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, 2000.

[38] Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S., "Making Gnutella-like P2P systems scalable," in *ACM SIGCOMM*, (Karlsruhe, Germany), ACM, August 2003.

[39] Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K., and Tierney, B., "Giggle: a framework for constructing scalable replica location services," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, (Los Alamitos, CA, USA), pp. 1–17, IEEE Computer Society Press, 2002.

[40] Chow, C.-Y., Mokbel, M. F., and Liu, X., "A peer-to-peer spatial cloaking algorithm for anonymous location-based service," in *GIS '06: Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, (New York, NY, USA), pp. 171–178, ACM, 2006.

[41] Chu, Y.-H., Rao, S. G., and Zhang, H., "A case for end system multicast," in *ACM SIGMETRICS 2000*, pp. 1–12, ACM, 2000.

[42] Coan, B. A., Oki, B. M., and Kolodner, E. K., "Limitations on database availability when networks partition," in *PODC '86: Proceedings of the fifth*

annual ACM symposium on Principles of distributed computing, (New York, NY, USA), pp. 187–194, ACM, 1986.

[43] Cohen, E. and Shenker, S., "Replication strategies in unstructured peer-to-peer networks," 2002. in The ACM SIGCOMM'02 Conference, August 2002.

[44] Cohen, E. I., King, G. M., and Brady, J. T., "Storage hierarchies," *IBM Syst. J.*, vol. 28, no. 1, pp. 62–76, 1989.

[45] Cuenca-Acuna, F. M., Martin, R. P., and Nguyen, T. D., "Autonomous Replication for High Availability in Unstructured P2P Systems," in *The 22nd IEEE Symposium on Reliable Distributed Systems (SRDS-22)*, IEEE Press, Oct. 2003.

[46] Dabek, F., Brunskill, E., Kaashoek, M. F., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H., "Building peer-to-peer systems with Chord, a distributed lookup service," pp. 81–86, 2001.

[47] Dabek, F., Cox, R., Kaashoek, F., and Morris, R., "Vivaldi: A decentralized network coordinate system," in *ACM SIGCOMM*, (Portland, Oregon, USA), ACM, August 2004.

[48] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I., "Wide-area cooperative storage with CFS," in *ACM SOSP*, 2001.

[49] Dasgupta, K., Ghosal, S., Jain, R., Sharma, U., and Verma, A., "Qosmig: Adaptive rate-controlled migration of bulk data in storage systems," *Data Engineering, International Conference on*, vol. 0, pp. 816–827, 2005.

[50] Davidson, S. B., Garcia-Molina, H., and Skeen, D., "Consistency in a partitioned network: a survey," *ACM Comput. Surv.*, vol. 17, no. 3, pp. 341–370, 1985.

[51] Douceur, J., "The Sybil attack," in *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, 2002.

[52] Failures, M. C., "Glacier: Highly durable, decentralized storage despite massive correlated failures."

[53] Fiege, L., Gartner, F. C., Kasten, O., and Zeidler, A., "Supporting mobility in content-based publish/subscribe middleware," in *Middleware*, 2003.

[54] Francis, P., "Yoid: Extending the multicast internet architecture." 1999.

[55] Ganesan, P., Bawa, M., and Garcia-Molina, H., "Online balancing of range-partitioned data with applications to peer-to-peer systems," in *VLDB*, 2004.

[56] Ganesan, P., Yang, B., and Molina, H. G., "One torus to rule them all: Multi-dimensional queries in p2p systems," in *WebDB*, 2004.

[57] Gedik, B. and Liu, L., "PeerCQ: A scalable and self-configurable peer-to-peer information monitoring system," Tech. Rep. GIT-CC-02-32, Georgia Institute of Technology, 2002.

[58] Gedik, B. and Liu, L., "PeerCQ: A decentralized and self-configuring peer-to-peer information monitoring system," in *International Conference on Distributed Computing Systems*, 2003.

[59] Gedik, B. and Liu, L., "Peercq: A decentralized and self-conuring peer-to-peer information monitoring system," in *Proceedings of ICDCS*, 2003.

[60] Gedik, B. and Liu, L., "Reliable peer-to-peer information monitoring through replication," in *Symposium on Reliable Distributed Systems*, 2003.

[61] Gedik, B. and Liu, L., "Reliable peer-to-peer information monitoring through replication," in *Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2003.

[62] Gedik, B. and Liu, L., "Mobieyes: A distributed location monitoring service using moving location queries," *IEEE Transactions on Mobile Computing*, 2005.

[63] Ghodsi, A., Alima, L. O., and Haridi, S., "Symmetric replication for structured peer-to-peer systems," in *DBISP2P*, pp. 74–85, 2005.

[64] Gnutella, "The gnutella home page." http://gnutella.wego.com/, 2002.

[65] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I., "Load balancing in dynamic structured P2P systems," in *Proc. IEEE INFOCOM*, (Hong Kong), 2004.

[66] Guerraoui, R. and Shiper, A., "Software-based replication for fault tolerance," *IEEE Computer - Special Issue on Fault Tolerance*, vol. 30, pp. 68–74, 1997.

[67] Gueye, B., Ziviani, A., Crovella, M., and Fdida, S., "Constraint-based geolocation of internet hosts," in *Proceedings of Internet Measurement Conference*, pp. 288–293, 2004.

[68] Gummadi, K., Dunn, R., Saroiu, S., Gribble, S., Levy, H., and Zahorjan, J., "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, Oct. 2003.

[69] hua Chu, Y., Rao, S. G., Seshan, S., and Zhang, H., "Enabling conferencing applications on the internet using an overlay multicast architecture," in *ACM SIGCOMM 2001*, (San Diago, CA), ACM, Aug.

[70] Huang, Y. and Garcia-Molina, H., "Publish/subscribe in a mobile environment," in *MobiDE*, 2001.

[71] Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, M. F., and O'Toole, Jr., J. W., "Overcast: Reliable multicasting with an overlay network," in *Proceedings of Symposium on Operating System Design and Implementation (OSDI)*, pp. 197–212, 2000.

[72] Kalnis, P., Ng, W. S., Ooi, B. C., Papadias, D., and Tan, K.-L., "An adaptive peer-to-peer network for distributed caching of olap results," in *ACM SIGMOD*, 2002.

[73] Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., and Panigrahy, R., "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *ACM Symposium on Theory of Computing Author Index*, pp. 654–663, May 1997.

[74] Katzan, Jr., H., "Storage hierarchy systems," in *AFIPS '71 (Spring): Proceedings of the May 18-20, 1971, spring joint computer conference*, (New York, NY, USA), pp. 325–336, ACM, 1971.

[75] Kommareddy, C., Shankar, N., and Bhattacharjee, B., "Finding close friends on the Internet," in *Proceedings of IEEE ICNP*, 2001.

[76] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B., "Oceanstore: an architecture for global-scale persistent storage," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 190–201, 2000.

[77] Liu, B., Lee, W.-C., and Lee, D. L., "Supporting complex multi-dimensional queries in p2p systems," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, 2005.

[78] Liu, L., Pu, C., and Tang, W., "Continual queries for internet scale event-driven information delivery," *IEEE Transactions on Knowledge and Data Engineering*, pp. 610–628, July/August 1999.

[79] Liu, L., Pu, C., and Tang, W., "Detecting and delivering information changes on the web," in *International Conference on Information and Knowledge Management*, 2000.

[80] Liu, L., Pu, C., and Tang, W., "Continual queries for internet scale event-driven information delivery," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 11, pp. 610 –628, jul/aug 1999.

[81] Liu, L., Pu, C., and Tang, W., "WebCQ: Detecting and delivering information changes on the web," in *CIKM*, pp. 512–519, 2000.

[82] Liu, L., Tang, W., Buttler, D., and Pu, C., "Information monitoring on the web: A scalable solution," *World Wide Web Journal*, 2003.

[83] Lu, C., Alvarez, G. A., and Wilkes, J., "Aqueduct: Online data migration with performance guarantees," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, (Berkeley, CA, USA), p. 21, USENIX Association, 2002.

[84] Luo, J. and Hubaux, J.-P., "A survey of inter-vehicle communication," Tech. Rep. IC/2004/24, School of Computer and Communication Sciences, EPFL, 2004.

[85] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S., "Search and replication in unstructured peer-to-peer networks," in *ICS '02: Proceedings of the 16th international conference on Supercomputing*, (New York, NY, USA), pp. 84–95, ACM, 2002.

[86] Maihöfer, C., "A survey on geocast routing protocols," *IEEE Communications Surveys and Tutorials*, vol. 6, no. 2, 2004.

[87] Maxion, R. A. and Reeder, R. W., "Improving user-interface dependability through mitigation of human error," *Int. J. Hum.-Comput. Stud.*, vol. 63, pp. 25–50, July 2005.

[88] Merugu, S. and et al, "p-sim: A simulator for peer-to-peer networks," in *MASCOTS*, 2003.

[89] Muhl, G., Ulbrich, A., Herrmann, K., and Weis, T., "Disseminating information to mobile clients using publish-subscribe," *IEEE Internet Computing*, vol. 08, no. 3, pp. 46–53, 2004.

[90] Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B., "Ivy: A read/write peer-to-peer file system," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[91] Nagaraja, K., Oliveira, F., Bianchini, R., Martin, R. P., and Nguyen, T. D., "Understanding and dealing with operator mistakes in internet services," in *In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI 04*, 2004.

[92] Napster, "http://www.napster.com," July 2006.

[93] Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., and Rowstron, A., "Migrating server storage to ssds: analysis of tradeoffs," in *EuroSys '09: Proceedings of the fourth ACM european conference on Computer systems*, (New York, NY, USA), pp. 145–158, ACM, 2009.

[94] Pandurangan, G., Raghavan, P., and Upfal, E., "Building low-diameter p2p networks," in *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, 2001.

[95] Paris, J., "Evaluating the impact of network partitions on replicated data availability," 1992.

[96] Pendarakis, D., Shi, S., Verma, D., and Waldvogel, M., "ALMI: An application level multicast infrastructure," in *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, pp. 49–60, 2001.

[97] Popek, G. J., Guy, R. G., Page, Jr., T. W., and Heidemann, J. S., "Replication in Ficus distributed file systems," in *IEEE Computer Society Technical Committee on Operating Systems and Application Environments Newsletter*, vol. 4, pp. 24–29, IEEE Computer Society, 1990.

[98] Prabhakaran, V., Rodeheffer, T. L., and Zhou, L., "Transactional flash," *Proceedings of the 8th USENIX Symposium on Operating System Design and Implementation*, December 2008.

[99] Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., and Stoica, I., "Load balancing in structured p2p systems," 2003.

[100] Rao, A., Lakshminarayanan, K., Surana, S., and title =, R. K.

[101] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., "A scalable content addressable network," in *Proceedings of SIGCOMM*, ACM, 2001.

[102] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S., "Application-level multicast using content-addressable networks," *Lecture Notes in Computer Science*, vol. 2233, 2001.

[103] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S., "Topologically-aware overlay construction and server selection," in *Proceedings of IEEE INFOCOM*, 2002.

[104] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J., "Handling churn in a dht," in *In Proceedings of the USENIX Annual Technical Conference*, 2004.

[105] Ripeanu, M., Foster, I., and Iamnitchi, A., "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[106] Rodrigues, L., Arajo, F., and Arajo, F., "Geopeer: A location-aware peer-to-peer system," tech. rep., In The 3rd IEEE International Conference on Network Computing and Applications (NCA 04, 2003.

[107] Rowstron, A., Kermarrec, A., Castro, M., and Druschel, P., "Scribe: The design of a large-scale event notification infrastructure," in *International Workshop on Networked Group Communication*, 2001.

[108] Rowstron, A. and Druschel, P., "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–??, 2001.

[109] Saroiu, S., Gummadi, P. K., and Gribble, S. D., "A measurement study of peer-to-peer file sharing systems," Tech. Rep. UW-CSE-01-06-02, University of Washington, 2001.

[110] Saroiu, S., Gummadi, P., and Gribble, S., "A measurement study of Peer-to-Peer file sharing systems," in *Proceedings of MMCN*, (San Jose, CA), August 2002.

[111] Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., and Steere, D. C., "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 447–459, 1990.

[112] Scheuermann, P., Weikum, G., and Zabback, P., "Adaptive load balancing in disk arrays," in *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO*, pp. 345–360, 1993.

[113] Schlichting, R. D. and Schneider, F. B., "Fail-stop processors: An approach to designing fault-tolerant computing systems," *Computer Systems*, vol. 1, no. 3, pp. 222–238, 1983.

[114] Shafaat, T. M., Ghodsi, A., and Haridi, S., "Handling network partitions and mergers in structured overlay networks," in *P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing (P2P 2007)*, (Washington, DC, USA), pp. 132–139, IEEE Computer Society, 2007.

[115] Spence, D., Crowcroft, J., Hand, S., and Harris, T., "Location based placement of whole distributed systems," in *CoNEXT '05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, (New York, NY, USA), pp. 124–134, ACM, 2005.

[116] Srivatsa, M. and Liu, L., "Vulnerabilities and security threats in structured overlay networks: A quantitative analysis," in *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, 2004.

[117] Srivatsa, M. and Liu, L., "Securing Publish-Subscribe overlay services with EventGuard," in *Proceedings of ACM Computer and Communication Security (CCS 2005)*, 2005.

[118] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings SIGCOMM*, pp. 149–160, ACM, 2001.

[119] Takemoto, D., Tagashira, S., and Fujita, S., "Distributed algorithms for balanced zone partitioning in content-addressable networks," in *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS'04)*, p. 377, 2004.

[120] Tang, L. and Crovella, M., "Virtual landmarks for the internet," in *Internet Measurement Conference*, pp. 143 – 152, 2003.

[121] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H., "Managing update conflicts in Bayou, a weakly connected replicated storage system," in *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15), Copper Mountain Resort, Colorado*, 1995.

[122] Triantafillou, P., Xiruhaki, C., Koubarakis, M., and Ntarmos, N., "Towards high performance peer-to-peer content and resource sharing systems," 2003.

[123] Vanthournout, K., Deconinck, G., and Belmans, R., "Building dependable peer-to-peer systems," 2004.

[124] Viana, A. C., de Amorim, M. D., Fdida, S., Viniotis, Y., and de Rezende, J. F., "Easily-managed and topology-independent location service for self-organizing networks," in *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, (New York, NY, USA), pp. 193–204, ACM, 2005.

[125] Vieira, M. and Madeira, H., "Recovery and performance balance of a cots dbms in the presence of operator faults," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, (Washington, DC, USA), pp. 615–626, IEEE Computer Society, 2002.

[126] Weatherspoon, H. and Kubiatowicz, J., "Erasure coding vs. replication: A quantitative comparison."

[127] Wilkes, J., Golding, R., Staelin, C., and Sullivan, T., "The HP AutoRAID hierarchical storage system," *ACM Trans. Comput. Syst.*, vol. 14, pp. 108–136, February 1996.

[128] Witt, H., "Exchanging information in decentralized location-based services," in *2005 2nd International Conference on Mobile Technology, Applications and Systems*, IEEE, 2005.

[129] Wouhaybi, R. H. and Campbell, A. T., "Phenix: Supporting resilient low-diameter peer-to-peer topologies," in *IEEE INFOCOM*, (Hong Kong, China), IEEE, March 2004.

[130] WWW, "The Gnutella RFC," *http://rfc-gnutella.sourceforge.net*, July 2006.

[131] Xu, Z., Mahalingam, M., and Karlsson, M., "Turning heterogeneity into an advantage in overlay routing," in *Proceedings of INFOCOM*, 2003.

[132] Xu, Z., Tang, C., and Zhang, Z., "Building topology-aware overlays using global soft-state," in *Proceedings of ICDCS*, 2003.

[133] Yu, H. and Vahdat, A., "Design and evaluation of a continuous consistency model for replicated services," pp. 305–318.

[134] Zegura, E. W., Calvert, K. L., and Bhattacharjee, S., "How to model an internetwork," in *IEEE Infocom*, vol. 2, pp. 594–602, IEEE, March 1996.

[135] Zhang, G., Chiu, L., Dickey, C., Liu, L., Muench, P., and Seshadri, S., "Automated lookahead data migration in ssd-enabled multi-tiered storage systems," 2010.

[136] Zhang, H., Goel, A., and Govindan, R., "Using the small-world model to improve freenet performance," in *Proc. IEEE Infocom, 2002. 14*, 2002.

[137] Zhang, J., Liu, L., and Pu, C., "Contructing a proximity-aware power law overlay network," in *Proceedings of the IEEE Global Telecommunications Conference, GLOBECOM 2005*.

[138] Zhang, J., Liu, L., Pu, C., and Ammar, M., "Reliable end system multicasting with a heterogeneous overlay network," Technical Report GIT-CERCS-04-19, CERCS, Georgia Institute of Technology, April 2004.

[139] Zhang, Z., Lin, S., Lian, Q., and Jin, C., "Repstore: A self-managing and self-tuning storage backend with smart bricks," *Autonomic Computing, International Conference on*, vol. 0, pp. 122–129, 2004.

[140] Zhao, B., Kubiatowicz, J., and Joseph, A., "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," tech. rep., U. C. Berkeley, 2002.

[141] Zhao, B. Y., Duan, Y., Huang, L., Joseph, A. D., and Kubiatowicz, J. D., "Brocade: Landmark routing on overlay networks," in *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*.

[142] Zhou, Y. and et al., "Adaptive reorganization of conherency-preserving dissemination tree for streaming data," in *ICDE*, 2006.

[143] Zhuang, S., Zhao, B., Joseph, A., Katz, R., and Kubiatowicz, J., "Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination," in *Proc. of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.